CS Exit Ethiopia

Questions & Worked Answers

Computer Science Exit Examination

278 complete questions

1 selected course | 18 November 2026

Selected courses

01. Software Engineering

https://csexitethiopia.com

abelasnake2003@gmail.com | Telegram: @Abechu7

Course contents

Each course begins on a new page. PDF bookmarks provide question, study-note section and revision-topic navigation.

Software Engineering 3

Reading the answers

Every question includes its answer, explanation, choice-by-choice contrasts and available evidence. Primary note proof quotes the study notes. Related reading and module passages are labelled separately; a module reference is not presented as primary note proof.

Software Engineering

Questions & Worked Answers

278 questions

Question 1 | Q00003

A company delivers registration first, then adds payments and reporting, each as a usable delivery. Which model specifically names this addition-of-functionality strategy, independent of whether earlier features are also refined?

A. Waterfall

B. Incremental

C. Spiral

D. Evolutionary

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Incremental

The delivery strategy adds usable capabilities in stages so that end users receive a valuable subset before the complete product is ready. Incremental development names that addition of functionality. Evolutionary development broadly emphasizes refinement and can overlap with increments, but does not specifically name the stated delivery distinction. Spiral is organized around risk-reduction cycles, and classical waterfall normally delays usable delivery until its planned development phases are complete.

Why each choice is right or wrong

A - Incorrect: Classical waterfall organizes sequential phases rather than prioritizing staged usable subsets.

B - Correct: Incremental development adds usable capability in successive deliveries.

C - Incorrect: Spiral names risk-driven cycles; those may contain increments without defining this delivery strategy.

D - Incorrect: Evolutionary refinement can overlap with increments, but the question asks the specific name for adding usable functionality.

Primary note proof

Study-note section SE04

> Choose incremental delivery when prioritized partial capability must reach users early. Smaller changes can be easier to debug and permit feedback, though architecture and integration still require planning. Iteration need not rebuild the whole system every time, and increments need

not be perfectly independent modules. Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Study-note section SE06

Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 22 | lines 8-12

This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
also provide a platform for evaluation by the user.

Software Engineering module | PDF page 20 | lines 18-22

the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
In principle, the waterfall model should only be used when the requirements are well understood
and unlikely to change radically during system development. However, the waterfall model

Question sources

MOE Computer Science Exit Exam · 2016 · Question 3

Question 2 | Q00010

If you want to test individual sub-systems of software, which phase of software testing is appropriate?

A. Integration testing

B. Unit testing

C. System testing

D. Acceptance testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: Unit testing

Unit testing tests isolated building blocks, including objects and subsystems; integration begins when individually tested components are combined.

Why each choice is right or wrong

A - Incorrect: Combines components rather than isolating a subsystem. See PDF p. 148, 144.

B - Correct: Directly tests an individual object or subsystem. See PDF p. 148, 144.

C - Incorrect: Tests components together as one system. See PDF p. 148, 144.

D - Incorrect: Checks the delivered system against the client agreement. See PDF p. 148, 144.

Primary note proof

Study-note section SE35

> Acceptance testing supports the customer/user's acceptance decision against agreed needs and criteria. Alpha typically happens in a controlled developer setting with selected users; beta uses external users in their environments. None of these levels is universally the longest. Unit is a level, while white-box is a technique; a unit can be tested using black-box cases too.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 148 | lines 19-26

> Unit testing focuses on the building blocks of the software system, that is, objects and subsystems.
> There are three motivations behind focusing on components. First, unit testing reduces the complexity of the overall test activities, allowing us to focus on smaller units of the system. Second, unit testing makes it easier to pinpoint and correct faults, given that few components are involved in the test. Third, unit testing allows parallelism in the testing activities; that is, each component can be tested independently of one another.
> The specific candidates for unit testing are chosen from the object model and the system decomposition of the system. In principle, all the objects developed during the development

Software Engineering module | PDF page 144 | lines 17-32

> Figure 6.2 depicts an overview of the testing activities:
> • Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
> • Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
> o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
> • System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:

o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Question sources

MOE Computer Science Exit Exam · 2016 · Question 10

Question 3 | Q00019

Which characteristic of requirements specifies that the requirement should be phrased so that there is one and only one interpretation for it?

A. Unambiguous

B. Correct

C. Feasible

D. Verifiable

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: Unambiguous

A single interpretation is the unambiguous property. Correctness, feasibility and verifiability answer different questions.

Why each choice is right or wrong

A - Correct: Rules out two or more interpretations. See PDF p. 68, 70.

B - Incorrect: Accuracy to the client's view does not guarantee a unique reading. See PDF p. 68, 70.

C - Incorrect: Realism concerns implementation within constraints. See PDF p. 68, 70.

D - Incorrect: Requires a repeatable test, not a unique interpretation alone. See PDF p. 68, 70.

Primary note proof

Study-note section SE11

Correct requirements reflect actual stakeholder needs. Complete requirements address the agreed scope, normal and exceptional behavior. Consistency excludes contradictions. Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable within constraints. Verifiability means objective evidence can check fulfillment. Replace “fast for most users” with a defined metric, workload and threshold.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 68 | lines 18-26

system specification. Requirement validation involves checking if the specification is correct, complete, consistent, unambiguous, and realistic. A specification is correct if it represents the client's view of the system (i.e., everything in the requirements model accurately represents an aspect of the system). It is complete if all possible scenarios through the system are described, including exceptional behavior (i.e., all aspects of the system are represented in the requirements
model). The system specification is consistent if it does not contradict itself. The system specification is unambiguous if exactly one system is defined (i.e., it is not possible to interpret the specification two or more different ways). Finally, it is realistic if the system can be implemented within constraints. These properties are illustrated with UML instance diagrams in

Software Engineering module | PDF page 70 | lines 3-16

3.2.5. Verifiability and Traceability
Two more desirable properties of a system specification are that it be verifiable and traceable. The
specification is verifiable if, once the system is built; a repeatable test can be designed to demonstrate that the system fulfills the requirement. For example, a mean time to failure of a hundred years for SatWatch would be difficult to achieve (assuming it is realistic in the first place).
The following requirements are additional examples of non-verifiable requirements:
• The product shall have a good user interface (good is not defined).
• The product shall be error free (requires large amount of resources to establish).
• The product shall respond to the user with 1 second for most cases ("most cases" is not defined).
A system specification is traceable if each system function can be traced to its corresponding set of requirements. Traceability is not a constraint on the content of the specification, but rather, on its organization. Traceability facilitates the development of tests and the systematic validation of the design against the requirements.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 19

Computer Science Exit Exam practice collection · Question 12

Question 4 | Q00082

Which option most completely summarizes the listed observable behaviors that functional requirements may specify?

A. specifies the tasks the program must complete

B. Specifies observable tasks the program must not perform

C. Specifies required observable responses to invalid input

D. All of the mentioned

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: All of the mentioned

Functional requirements describe observable services and behavior, including what must happen, what is prohibited and how the system responds to errors. Each individual choice gives one of these forms. The all-of-the-mentioned option is therefore the complete summary. Selecting only A, B or C would omit other valid functional obligations; negative wording alone does not make a requirement nonfunctional.

Why each choice is right or wrong

A - Incorrect: Required tasks are functional, but A alone omits the other listed forms.

B - Incorrect: Prohibited tasks can be functional, but B alone is incomplete.

C - Incorrect: Required error responses are functional, but C alone is incomplete.

D - Correct: All three listed observable behavior forms can be functional requirements.

Primary note proof

Study-note section SE09

> A functional requirement states an observable service or behavior, including reactions to inputs, errors and forbidden actions. “When the sensor detects intrusion, send an alert” is functional. “Only a manager may approve refunds” also specifies permitted/prohibited behavior. A negative statement is not automatically nonfunctional.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 65 | lines 19-23

> 3.2.1. Functional Requirements
> Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. For example, the following is an example of functional requirements for SatWatch, a watch that resets itself without user intervention:

Question sources

MOE Computer Science Exit Exam · 2016 · Question 82

Question 5 | Q00083

A team has little experience with a project's unfamiliar technology. Major feasibility risks must be evaluated before each further commitment, and experienced risk-analysis support is available. Which lifecycle is selected specifically by that risk-driven criterion?

A. Iterative Enhancement Model

B. RAD

C. Spiral

D. Waterfall

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Spiral

Spiral organizes each cycle around objectives, alternatives, risk assessment and reduction before further commitment. That directly addresses the specified major feasibility risks and supported risk-analysis capability. Iterative enhancement can aid learning but does not make risk assessment its defining organizer; RAD needs rapid construction and skilled support. Classical waterfall fits stable requirements and familiar technology, which the scenario does not supply.

Why each choice is right or wrong

A - Incorrect: Iteration supports feedback but is not defined by risk-driven commitments.

B - Incorrect: RAD emphasizes rapid construction rather than the specified risk cycle.

C - Correct: Spiral explicitly organizes commitment around major risk reduction.

D - Incorrect: Unfamiliar technology and unresolved risks do not fit the stated classical waterfall rationale.

Primary note proof

Study-note section SE02

> Select a process using requirement stability, technical risk, delivery urgency, stakeholder availability, team skills and assurance needs. "Team has little experience" alone does not uniquely identify a process. A risk-driven cycle, prototype or carefully planned staged process may each help, depending on the actual uncertainty. No process eliminates the need for testing or documentation.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 20 | lines 18-22

> the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
> In principle, the waterfall model should only be used when the requirements are well understood
> and unlikely to change radically during system development. However, the waterfall model

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
> follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 83

MOE Computer Science Exit Exam · 2016 · Question 95

Question 6 | Q00084

In which step of SDLC actual programming of software code is done?

A. Development and Documentation

B. Maintenance and Evaluation

C. Design

D. Analysis

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Development and Documentation

Programming translates the design into source code in implementation. A corresponds to the development stage of implementation and unit testing.

Why each choice is right or wrong

A - Correct: Development is the option corresponding to producing source code; documentation accompanies the process. See PDF p. 18, 18, 19.

B - Incorrect: Maintenance modifies an operational product. See PDF p. 18, 18, 19.

C - Incorrect: Design derives implementation structure rather than source code. See PDF p. 18, 18, 19.

D - Incorrect: Analysis establishes and organizes requirements. See PDF p. 18, 18, 19.

Primary note proof

Study-note section SE02

A common lifecycle names requirements, design, implementation, testing, deployment and maintenance. Analysis clarifies the problem; system design chooses architecture; object/detail design specifies interfaces and algorithms; implementation writes code; testing evaluates behavior; deployment puts it into service. These activities may overlap and repeat.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 18 | lines 28-32

3. Implementation and unit testing: During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification. The purpose of the coding phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is
implemented as a program module. The end-product of this phase is a set of program modules that

Software Engineering module | PDF page 18 | lines 13-20

System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
Software design involves identifying and describing the fundamental software system abstractions
and their relationships. The goal of the design phase is to transform the requirements specified in
the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Software Engineering module | PDF page 19 | lines 24-29

5. Operation and maintenance: Normally (although not necessarily), this is the longest life cycle phase. Maintenance of a typical software product requires much more than the effort necessary to
develop the product itself. The system is installed and put into practical use. Maintenance involves
correcting errors which were not discovered in earlier stages of the life cycle, improving the

implementation of system units and enhancing the system ‘s services as new requirements are discovered.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 84

MOE Computer Science Exit Exam · 2016 · Question 96

Question 7 | Q00094

Which offered requirement is nonfunctional, specifying a measurable quality rather than required or prohibited observable tasks?

A. specifies the tasks the program must complete

B. Specifies observable tasks the program must not perform

C. Specifies the required error response to invalid input

D. Search shall respond within two seconds under 100 concurrent users.

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: Search shall respond within two seconds under 100 concurrent users.

The response-time limit under a defined concurrent-user workload is a measurable performance requirement, a nonfunctional quality constraint. Required services, prohibited actions and specified error responses are all forms of observable functional behavior. Their positive or negative wording does not determine the category. D instead constrains how well the search service must perform, making it the unique nonfunctional example.

Why each choice is right or wrong

A - Incorrect: Required observable tasks are functional.

B - Incorrect: Prohibited observable tasks can be functional.

C - Incorrect: An observable error response specifies functional behavior.

D - Correct: The time threshold and workload constrain performance quality.

Primary note proof

Study-note section SE09

A functional requirement states an observable service or behavior, including reactions to inputs, errors and forbidden actions. “When the sensor detects intrusion, send an alert” is functional. “Only a manager may approve refunds” also specifies permitted/prohibited behavior. A negative statement is not automatically nonfunctional.

Study-note section SE10

"Search responds within two seconds under 100 concurrent users" is measurable performance. "New users finish checkout within one minute without help" is usability. "Use Java" is a technology/pseudo-requirement, often separately classified as a system constraint. Dependability includes reliability, safety and robustness; portability is not its usual central category. Performance degradation at peak load primarily concerns performance, not automatically reliability.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 65 | lines 19-23

3.2.1. Functional Requirements
Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. For example, the following is an example of functional requirements for SatWatch, a watch that resets itself without user intervention:

Question sources

MOE Computer Science Exit Exam · 2016 · Question 94

Question 8 | Q00098

Which of the following is the understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc.?

A. Software Design

B. Feasibility Study

C. Requirement Gathering

D. System Analysis

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Feasibility study.

SE08 asks whether a proposal is worth doing and deliverable, considering existing limitations, impacts on organization and personnel, benefits, costs, skills and integration risk. That matches the stem's preliminary impact and constraint investigation. System analysis studies the detailed problem and behavior and informs feasibility, but the broad early viability assessment is the more specific match here.

Why each choice is right or wrong

A - Incorrect: Design chooses solution structures rather than primarily assessing preliminary viability and organizational impacts.

B - Correct: Feasibility investigates existing limitations, impacts and whether the proposal is worthwhile and deliverable.

C - Incorrect: Gathering needs is useful input but does not by itself cover the full viability assessment.

D - Incorrect: System analysis examines detailed problem behavior; feasibility is the more specific preliminary impact/viability activity.

Primary note proof

Study-note section SE08

> Feasibility asks whether a proposal is worth doing and can be delivered within technical, economic, operational and schedule constraints. Examine existing limitations, organizational impacts, benefits, cost, available skills and integration risk. System analysis studies the problem and needed behavior in more detail; the activities inform each other.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 25 | lines 15-18

> There are four main activities in the requirements engineering process:
> 1. Feasibility study: Feasibility shows viability and practicality of a software, project or idea. An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-

Software Engineering module | PDF page 26 | lines 3-5

> effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.

Software Engineering module | PDF page 100 | lines 11-24

> Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users
> Figure 4.1 Products of requirements elicitation and analysis (UML activity diagram).
> Althoughtheanalysismodelmaynotbeunderstandabletotheusersandtheclient, it helps developers verify the system specification produced during requirements elicitation. There is a natural tendency for users and developers to postpone difficult decisions until later in the project. A decision may be difficult because of lack of domain knowledge, lack of technological knowledge, or simply because of disagreements among users and developers. Postponing decisions enables the
> project to move on smoothly and avoid confrontation with reality or peers. Unfortunately, difficult
> decisions will eventually need to be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation. Translating a system specification in to a formal or semi-formal model forces developers to identify and resolve difficult issues early

in
the development.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 98

Question 9 | Q00120

For a software product developed to meet a client’s requirements, which choice most completely describes its scope?

A. Software product refers any software created to meet the client's requirements.

B. Software product includes design and test documentation.

C. Software product includes requirement specifications, and user manuals.

D. All of the above

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer D: All of the above

SE01 explicitly includes executable code plus requirements, designs, test assets, configuration and user documentation in a software product. A describes the implemented software, while B and C name relevant accompanying artifacts. All three descriptions are therefore compatible; none limits the product to code alone. D is the inclusive answer among these options.

Why each choice is right or wrong

A - Incorrect: True as one component, but incomplete as the requested full selection. Software built to meet client requirements forms part of the product.

B - Incorrect: True as one component, but incomplete as the requested full selection. Relevant design and testing documentation are included product artifacts.

C - Incorrect: True as one component, but incomplete as the requested full selection. Requirements specifications and user manuals are relevant product documentation.

D - Correct: All three descriptions are compatible with the complete software-product scope.

Primary note proof

Study-note section SE01

Software engineering applies systematic methods to development, operation and maintenance. A software product includes executable code and relevant requirements, designs, test assets, configuration and user documentation. A process defines activities, ordering, roles, products and entry/exit conditions. Its four broad activities are specification, design/implementation, validation and evolution.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 89 | lines 17-24

> Assigning a role to a participant usually translates into assigning the responsibility of a specific work product. A work product is an artifact that is produced during the development, such as an internal document for other project participants or a deliverable to the customer.
> An analyst is responsible for a section of the RAD describing the functionality of the system. A system architect is responsible for the SDD describing the system's high-level architecture. A project manager is responsible for the SPMP describing the development processes. Finally, the system and its accompanying documentation also constitute a set of work products that are delivered to the client.

Software Engineering module | PDF page 83 | lines 12-25

> 3.4.4. Documenting Requirements Elicitation
> The results of the requirements elicitation activity and the analysis activity are documented in the
> Requirements Analysis Document (RAD). This document completely describes the system in terms of functional and nonfunctional requirements and serves as a contractual basis between the
> client and the developers. The audience for the RAD includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design). The first part of the document, including use cases and nonfunctional requirements, is written during requirements
> elicitation. The formalization of the specification in terms of object models is written during analysis.
> The first section of the RAD is an Introduction. Its purpose is to provide a brief overview of the function of the system and the reasons for its development, its scope, and references to the development context (e.g., related problem statement, references to existing systems, feasibility studies). The introduction also includes the objectives and success criteria of the project.

Software Engineering module | PDF page 130 | lines 3-11

> 6.3. DOCUMENTING SYSTEM DESIGN
> System design is documented in the System Design Document (SDD). It describes design goals set by the project, the subsystem decomposition (with UML class diagrams), the hardware/software mapping (with UML deployment diagrams), the data management, the access control, control flow mechanisms, and boundary conditions. The SDD is used to define interfaces between teams of developers and as reference when architecture-level decisions need to be revisited. The audience for the SDD includes the project management, the system architects (i.e., the developers who participate in the system design), and the developers who design and implement each subsystem. The following is an example template for a SDD:

Question sources

Addis Ababa University mock exam · Question 22

Question 10 | Q00121

One of the followings is type of requirements associated with quality features of a software.

A. Business requirements

B. Functional requirements requirements

C. Non-functional

D. User requirements

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Non-functional

Quality and user-visible constraints beyond functional behavior are nonfunctional requirements; user and business are viewpoints rather than this quality classification.

Why each choice is right or wrong

A - Incorrect: Business objectives do not identify the stated quality category. See PDF p. 66, 78.

B - Incorrect: Specifies provided interactions and services. See PDF p. 66, 78.

C - Correct: Matches nonfunctional quality constraints. See PDF p. 66, 78.

D - Incorrect: User requirements can contain several types, so this is broader than quality features. See PDF p. 66, 78.

Primary note proof

Study-note section SE10

> Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning. Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 66 | lines 14-18

> 3.2.2. Non-Functional and Pseudo-Requirements
> Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Nonfunctional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands)
> or accuracy (i.e., how precise are the system’s numerical answers). The following are the

Software Engineering module | PDF page 78 | lines 12-30

3.3.7. Identifying Non-Functional Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related
to the functional behavior of the system. Nonfunctional requirements span a number of issues, from user interface look and feel to response time requirements to security issues. Nonfunctional
requirements are defined at the same time as functional requirements are, because they have as much impact on the development and cost of the system.
For example, consider a mosaic display that an air traffic controller uses to track planes. A mosaic
display system complies data from a series of radar and databases (hence the term “mosaic”) into
a summary display indicating all aircraft in a certain area, including their identification, speed, and
altitude. The number of aircraft such a system can display constrains the performance of the air traffic controller and the cost of the system. If the system can only handle a few aircraft simultaneously, the system cannot be used at busy airports. On the other hand, a system able to handle a large number of aircraft is more costly and more complex to build.
Nonfunctional requirements can be elicited by investigating the following issues.
• User interface and human factors. What kind of interface should the system provide? What is the level of expertise of the users?
• Documentation. What level of document is required? Should only user documentation be provided? Should there be technical documentation for maintainers? Should the development process be documented?

Question sources

Addis Ababa University mock exam · Question 23

Question 11 | Q00122

A type of requirement that describes why the software project is needed and used to define the objective of the software is .

A. Business requirements

B. Functional requirements

C. Non-functional requirements

D. User requirements

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: Business requirements.

SE08 distinguishes the business reason and objectives of a project from user needs and detailed system obligations. The question asks why the project is needed and what objectives it serves, so business requirements match directly. Functional requirements describe behavior, while nonfunctional requirements describe qualities or constraints; neither primarily states the project's business justification.

Why each choice is right or wrong

A - Correct: Business requirements state the project's purpose, justification and intended organizational goals.

B - Incorrect: Functional requirements specify observable system services rather than the overall business reason for the project.

C - Incorrect: Nonfunctional requirements specify qualities or constraints rather than why the project exists.

D - Incorrect: User requirements describe stakeholder needs at a high level rather than the project's business justification.

Primary note proof

Study-note section SE08

> Business requirements explain why the project exists and its goals. User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements. A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 83 | lines 12-25

> 3.4.4. Documenting Requirements Elicitation
> The results of the requirements elicitation activity and the analysis activity are documented in the
> Requirements Analysis Document (RAD). This document completely describes the system in terms of functional and nonfunctional requirements and serves as a contractual basis between the
> client and the developers. The audience for the RAD includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design). The first part of the document, including use cases and nonfunctional requirements, is written during requirements
> elicitation. The formalization of the specification in terms of object models is written during analysis.
> The first section of the RAD is an Introduction. Its purpose is to provide a brief overview of the function of the system and the reasons for its development, its scope, and references to the

development context (e.g., related problem statement, references to existing systems, feasibility studies). The introduction also includes the objectives and success criteria of the project.

Software Engineering module | PDF page 26 | lines 10-17

3. Requirement's specification: Requirements specification is the activity of translating the information gathered during the analysis activity in to a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are
abstract statements of the system requirements for the customer and end-user of the system; system
requirements are a more detailed description of the functionality to be provided.
4. Requirement's validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered.
It must then be modified to correct these problems.

Question sources

Addis Ababa University mock exam · Question 24

Question 12 | Q00123

Which of the following is used to model the functionality of software?

A. Use case diagram

B. Class diagram

C. E-R diagram

D. Deployment diagram

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer A: Use case diagram

Use-case diagrams model external roles and their goals or interactions with the system. This is the functional view requested. Class diagrams instead emphasize static classes, features and relationships. Entity-relationship diagrams organize entity types, attributes and data relationships. Deployment diagrams map software artifacts to execution nodes. Each can contribute to understanding a system, but only the use-case view directly presents the actor-goal functionality in this comparison.

Why each choice is right or wrong

A - Correct: Use-case diagrams show external actors and the system goals they use.

B - Incorrect: Class diagrams describe static types and their relationships.

C - Incorrect: ER diagrams describe entity types, attributes and data relationships.

D - Incorrect: Deployment diagrams map software artifacts onto execution nodes.

Primary note proof

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Study-note section SE-S03

> An entity-relationship diagram models entity types, their attributes and the relationships between them, commonly for data design. Its focus differs from a use-case diagram of external roles and goals, a class diagram of classes and their features, or a deployment diagram mapping artifacts to execution nodes.

Study-note section SE24

> A component diagram shows components/interfaces/dependencies; a deployment diagram places software artifacts on nodes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

> Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
> Use cases describe the behavior of the system as seen from an actor’s point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
> use case to access the system functionality. The use case can then initiate other use cases and gather
> more information from the actors. When actors and use cases exchange information, they are said
> to communicate. We will see later that we represent these exchanges with communication relationships.

Software Engineering module | PDF page 51 | lines 14-24

> 2.3.2. Class Diagrams
> Classes and objects
> Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behavior of a set of objects. Objects are entities that encapsulate state and behavior. Each object has an identity: It can be referred individually and is distinguishable from other objects.
> In UML, classes and objects are depicted by boxes including three compartments. The top

compartment displays the name of the class or object. The center compartment displays its attributes, and the bottom compartment displays its operations. The attribute and operation compartment can be omitted for clarity. Object names are underlined to indicate that they are instances. By convention, class names start with an uppercase letter. Objects in object diagrams

Software Engineering module | PDF page 130 | lines 3-11

6.3. DOCUMENTING SYSTEM DESIGN
System design is documented in the System Design Document (SDD). It describes design goals set by the project, the subsystem decomposition (with UML class diagrams), the hardware/software mapping (with UML deployment diagrams), the data management, the access control, control flow mechanisms, and boundary conditions. The SDD is used to define interfaces between teams of developers and as reference when architecture-level decisions need to be revisited. The audience for the SDD includes the project management, the system architects (i.e., the developers who participate in the system design), and the developers who design and implement each subsystem. The following is an example template for a SDD:

Question sources

Addis Ababa University mock exam · Question 25

Question 13 | Q00124

Which of the following refers to the deployment platform? i

A. Use case

B. Tier

C. Layer

D. Component

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Tier.

SE23 explicitly separates logical abstraction from deployment separation. A layer organizes responsibilities logically, whereas a tier describes their deployment separation or platform arrangement. A use case models an actor goal, and a component is a software building block. Neither is the term for the deployment distinction asked for here.

Why each choice is right or wrong

A - Incorrect: A use case models a goal or interaction rather than the deployment platform separation.

B - Correct: A tier identifies deployment separation, contrasting with logical layering.

C - Incorrect: A layer is a logical abstraction level and need not occupy a distinct deployment platform.

D - Incorrect: A component is a software element and does not itself denote a deployment tier.

Primary note proof

Study-note section SE23

A layer is a logical abstraction level; a tier is a deployment separation. A closed layered architecture permits calls only to the immediately lower layer; an open one permits approved skipping. For top-to-bottom A,B,C,D, A→B obeys closed layering, A→C skips, and lower-to-upper calls violate the simple dependency rule unless explicitly mediated by a defined mechanism.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 130 | lines 3-11

6.3. DOCUMENTING SYSTEM DESIGN
System design is documented in the System Design Document (SDD). It describes design goals set by the project, the subsystem decomposition (with UML class diagrams), the hardware/software mapping (with UML deployment diagrams), the data management, the access control, control flow mechanisms, and boundary conditions. The SDD is used to define interfaces between teams of developers and as reference when architecture-level decisions need to be revisited. The audience for the SDD includes the project management, the system architects (i.e., the developers who participate in the system design), and the developers who design and implement each subsystem. The following is an example template for a SDD:

Software Engineering module | PDF page 67 | lines 3-8

Pseudo-requirements are requirements imposed by the client that restrict the implementation of the system. Typical pseudo-requirements are the implementation language and the platform on which the system is to be implemented. For life-critical developments, pseudo-requirements often
include process and documentation requirements (e.g., the use of a formal specification method, the complete release of all work products). Pseudo-requirements have usually no direct effect on the users' view of the system. The following are the pseudo functional requirements for SatWatch:

Question sources

Addis Ababa University mock exam · Question 26

Question 14 | Q00125

When requirements will change substantially and frequent early feedback is needed, which classical process organization is least accommodating among these choices?

A. Waterfall

B. Scrum

C. RAD

D. Agile

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Waterfall

Classical waterfall commits to planned phases and reviewed deliverables, making late feedback and changed requirements costly. Agile and Scrum plan and inspect work incrementally, allowing later content to adapt. RAD also uses rapid construction and user participation to obtain feedback. The comparison therefore favors waterfall as the least accommodating under the stated conditions, without claiming that a waterfall project is physically unable to authorize any change.

Why each choice is right or wrong

A - Correct: Late feedback and phase commitments make major requirement changes costly in classical waterfall.

B - Incorrect: Scrum inspects usable increments and adapts later work within its planning framework.

C - Incorrect: RAD uses rapid construction and user participation to obtain feedback.

D - Incorrect: Agile explicitly values responding to changing needs.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE05

> Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Study-note section SE07

A prototype explores requirements, usability or technical feasibility. A throwaway prototype is discarded after learning; an evolutionary prototype is deliberately refined toward production. A quick prototype may omit robustness, security or architecture, so do not deploy it unchanged without evaluation. Coding may occur during prototyping even if a diagram labels that activity construction rather than coding.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 20 | lines 18-22

the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
In principle, the waterfall model should only be used when the requirements are well understood
and unlikely to change radically during system development. However, the waterfall model

Software Engineering module | PDF page 13 | lines 22-27

Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Question sources

Addis Ababa University mock exam · Question 27

Question 15 | Q00126

Which of the following aims to find errors?

A. Requirement Analysis

B. Design

C. Testing

D. None of the above

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer C: Testing

Testing systematically attempts to expose errors in the implementation; the other named activities primarily construct specifications and designs.

Why each choice is right or wrong

A - Incorrect: Structures requirements, although validation may reveal requirement defects. See PDF p. 140, 100, 18.

B - Incorrect: Constructs the solution design. See PDF p. 140, 100, 18.

C - Correct: Explicit planned fault detection activity. See PDF p. 140, 100, 18.

D - Incorrect: Testing is present, so none is false. See PDF p. 140, 100, 18.

Primary note proof

Study-note section SE34

A fault/defect is an underlying problem; an error may be an incorrect internal state; a failure is an externally observed departure from required service. Reliability is a product quality attribute; low cost or short development time are project outcomes. Tests increase confidence and reveal defects but cannot generally guarantee correctness over every possible input.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 140 | lines 27-32

7.1 OVERVIEW OF SOFTWARE QUALITY ASSURANCE
We define testing as the systematic attempt to find errors in a planned way in the implemented software. Contrast this definition with another commonly used definition that says that “testing is
the process of demonstrating that errors are not present.” The distinction between these two definitions is important. Our definition does not mean we simply demonstrate that the program does what it is intended to do. The explicit goal of testing is to demonstrate the presence of faults.

Software Engineering module | PDF page 100 | lines 11-24

Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users
Figure 4.1 Products of requirements elicitation and analysis (UML activity diagram).
Althoughtheanalysismodelmaynotbeunderstandabletotheusersandtheclient, it helps developers verify the system specification produced during requirements elicitation. There is a natural tendency for users and developers to postpone difficult decisions until later in the project. A decision may be difficult because of lack of domain knowledge, lack of technological knowledge, or simply because of disagreements among users and developers. Postponing decisions enables the
project to move on smoothly and avoid confrontation with reality or peers. Unfortunately, difficult
decisions will eventually need to be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation. Translating a system specification in to a formal or semi-formal model forces developers to identify and resolve difficult issues early in
the development.

Software Engineering module | PDF page 18 | lines 13-20

System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
Software design involves identifying and describing the fundamental software system abstractions
and their relationships. The goal of the design phase is to transform the requirements specified in
the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Question sources

Addis Ababa University mock exam · Question 28

Question 16 | Q00239

Which among the following focuses on the way the elements interact with each other at runtime during architectural design?

A. All choices are correct

B. Component-and-connector (C&C) structures

C. Allocation structures

D. Module decomposition structure

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Component-and-connector structures.

The runtime interaction of software elements is the subject of component-and-connector structures in SE23. Module structures instead organize the code, while allocation structures relate software to hardware, teams or environments. These are different architectural views, so the inclusive choice fails: only the C&C option specifically answers the runtime-interaction focus.

Why each choice is right or wrong

A - Incorrect: The alternatives address different architectural concerns, so not every choice focuses on runtime interaction.

B - Correct: C&C structures show runtime elements and the connectors through which they interact.

C - Incorrect: Allocation structures map software to hardware, teams or environments rather than primarily describing runtime exchanges.

D - Incorrect: Module decomposition shows code organization rather than the runtime interactions emphasized by the stem.

Primary note proof

Study-note section SE23

> A structure is the elements/relations themselves; a view represents selected aspects. Module structures show code organization, component-and-connector structures runtime interaction, and allocation structures map software to hardware, teams or environment. Processes/threads belong in concurrency views. Evaluate architectures against quality attributes and scenarios; processor allocation, decomposition and style selection can all be architectural concerns.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 121 | lines 6-15

> 6.1.4. Software architecture
> As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
> of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
> In this section, we describe a few sample architectures that can be used for different systems. This
> is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
> representative examples and refer the reader to the literature for more details.

Question sources

Mock exam 4 · Question 41

Question 17 | Q00240

Which option combines both explicitly specified testability mechanisms among A, C and D? Treat refactoring alone as behavior-preserving restructuring with no stated testability improvement.

A. Sand box

B. A and D

C. Refactoring

D. Limiting structural complexity

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: A and D

A sandbox can isolate execution, giving control over the test environment. Limiting structural complexity also supports testability by making behavior easier to control and observe. Refactoring preserves behavior while changing structure, but without a stated improvement it does not automatically supply either testability effect. The combination A and D therefore includes both explicitly supported mechanisms; the individual choices each give only one.

Why each choice is right or wrong

A - Incorrect: Sandbox isolation is valid but only one of the two mechanisms.

B - Correct: A and D include both explicitly supported mechanisms.

C - Incorrect: Unspecified refactoring need not improve testability.

D - Incorrect: Limited structural complexity is valid but only one of the two mechanisms.

Primary note proof

Study-note section SE27

> Variability can use selectable features, omission/inclusion and extension points; overloading supplies alternate signatures in code but is not every variability mechanism. Testability improves with controllable inputs, observable outputs, replaceable dependencies and limited structural complexity. A sandbox can isolate execution. Assess each design claim independently before evaluating a combined statement.

Study-note section SE22

> Logical cohesion groups similar-category operations, often selected by a flag; temporal cohesion groups operations done at the same time; functional cohesion focuses on one task. Splitting an unrelated three-purpose component improves modularity. If behavior is preserved, the restructuring is refactoring; the words are not mutually exclusive answers. Smaller modules can aid understanding, independent work, reuse and maintenance, but too many interfaces also create complexity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 127 | lines 10-28

> 6.2.1. Identifying design goals
> The definition of design goals is the first step of system design. It identifies the qualities that our system should focus on. Many design goals can be inferred from the nonfunctional requirements or from the application domain. Others will have to be elicited from the client. It is, however, necessary to state them explicitly such that every important design decision can be made consistently following the same set of criteria.
> For example, in the light of the nonfunctional requirements for MyTrip, we identify reliability and
> fault tolerance to connectivity loss as design goals. We then identify security as a design goal, as numerous drivers will have access to the same trip planning server. We add modifiability as a design goal, as we want to provide the ability for drivers to select a trip planning service of their

choice. The following box summarizes the design goals we identified.
In general, we can select design goals from a long list of highly desirable qualities. Tables 5-1-5-5 list a number of possible design criteria. These criteria are organized into five groups: performance, dependability, cost, maintenance, and end user criteria. Performance, dependability,
and end user criteria are usually specified in the requirements or inferred from the application domain. Cost and maintenance criteria are dictated by the customer and the supplier.
Performance criteria (Table 5-1) include the speed and space requirements imposed on the system. Should the system be responsive or should it accomplish a maximum number of tasks? Is memory space available for speed optimizations or should memory be used sparingly?

Software Engineering module | PDF page 133 | lines 3-15

transformation on subsets of a particular model. Typical activities include transforming n-ary associations into binary associations, implementing binary associations as references, merging two similar classes from two different subsystems into a single class, collapsing classes with no significant behavior into attributes, splitting complex classes into simpler ones, rearranging classes and operations to increase the inheritance and packaging. During restructuring, we address design goals such as maintainability, readability, and understandability of the system model.
• Optimization. Optimization activities address performance requirements of the system model. This includes changing algorithms to respond to speed or memory requirements, reducing multiplicities in associations to speed up queries, adding redundant associations for efficiency, rearranging execution orders, adding derived attributes to improve the access time to objects and opening up the architecture, that is, adding access to lower layers because of performance requirements.

Question sources

Mock exam 4 · Question 42

Question 18 | Q00241

Which choices explicitly provide architectural variability through selectable elements or defined extension interfaces? Select all that apply. Treat overloading as alternate code signatures only.

A. Overloading

B. Omission or inclusion of elements

C. Extension point

D. All of the above

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B, C: Omission or inclusion of elements; Extension point

Omitting or including selected elements permits configured architectural variation, and extension points define places where additional behavior can be supplied. Overloading by itself supplies alternate code signatures and does not establish either architectural mechanism. Thus B and C meet the specified architectural scope. The all-of-the-above choice includes that insufficient code-level case and therefore cannot be selected as a correct aggregate.

Why each choice is right or wrong

A - Incorrect: Alternate signatures alone do not define the specified architectural variability.

B - Correct: Selectable omission or inclusion changes architectural configuration.

C - Correct: Extension points explicitly support variation at defined boundaries.

D - Incorrect: It incorrectly includes overloading alone as an architectural mechanism.

Primary note proof

Study-note section SE27

> Variability can use selectable features, omission/inclusion and extension points; overloading supplies alternate signatures in code but is not every variability mechanism. Testability improves with controllable inputs, observable outputs, replaceable dependencies and limited structural complexity. A sandbox can isolate execution. Assess each design claim independently before evaluating a combined statement.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 121 | lines 6-15

> 6.1.4. Software architecture
> As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
> of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
> In this section, we describe a few sample architectures that can be used for different systems. This
> is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
> representative examples and refer the reader to the literature for more details.

Software Engineering module | PDF page 104 | lines 6-17

> 5.1.4. Generalization
> Generalization enables us to organize concepts in to hierarchies. There may be any number of intermediate levels in between, covering more-or-less generalized concepts (e.g., Low Priority Incident, Emergency, Disaster). Such hierarchies allow us to refer to many concepts precisely.

When
we use the term Incident, we mean all instances of all types of Incidents. When we use the term Emergency, we only refer to an Incident that requires an immediate response. This view of generalization stems from modeling.
In an object-oriented programming language, inheritance is a reusability technique. If a class Child
inherits from a class Parent, all the attributes and methods available on Parent are automatically available on Child. The class Child may add additional methods or override inherited methods, thus refining the class Parent. As in the case of generalization, the class at the top of the hierarchy
tends to be the most general one, whereas the leaves tend to be the most specialized.

Question sources

Mock exam 4 · Question 43

Question 19 | Q00242

Which choice most completely describes the stated advantages of the Factory Method pattern?

A. Allows the sub-classes to choose the type of objects to create

B. Promotes loose coupling

C. A and B

D. None of the above

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: A and B

SE27 describes Factory Method as letting subclasses choose a concrete product through a creation operation, reducing dependence on concrete constructors. Those are exactly the two benefits in A and B, so C is the inclusive answer. The reference is to Factory Method; a simple factory is related but does not necessarily delegate product choice to subclasses.

Why each choice is right or wrong

A - Incorrect: True as one component, but incomplete as the requested full selection. Factory Method lets subclasses select the concrete product through the creation operation.

B - Incorrect: True as one component, but incomplete as the requested full selection. Depending on the creation abstraction reduces coupling to concrete product constructors.

C - Correct: Both listed benefits hold for Factory Method, making this the inclusive answer.

D - Incorrect: The note explicitly supports both benefits, so none is false.

Primary note proof

Study-note section SE27

Factory Method lets subclasses choose a concrete product through a creation operation, reducing coupling to concrete constructors. A simple factory centralizes creation but is not identical to Factory Method. Singleton controls access to one instance; Observer notifies subscribers; Memento stores/restores state while hiding representation; Iterator traverses; Decorator adds behavior through wrapping. The pattern name is Memento, not Momento.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 120 | lines 6-12

6.1.3. Coupling and coherence
Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Software Engineering module | PDF page 104 | lines 6-17

5.1.4. Generalization
Generalization enables us to organize concepts in to hierarchies. There may be any number of intermediate levels in between, covering more-or-less generalized concepts (e.g., Low Priority Incident, Emergency, Disaster). Such hierarchies allow us to refer to many concepts precisely. When
we use the term Incident, we mean all instances of all types of Incidents. When we use the term Emergency, we only refer to an Incident that requires an immediate response. This view of generalization stems from modeling.
In an object-oriented programming language, inheritance is a reusability technique. If a class Child
inherits from a class Parent, all the attributes and methods available on Parent are automatically available on Child. The class Child may add additional methods or override inherited methods, thus refining the class Parent. As in the case of generalization, the class at the top of the hierarchy
tends to be the most general one, whereas the leaves tend to be the most specialized.

Question sources

Mock exam 4 · Question 44

Question 20 | Q00243

Which statements correctly distinguish architectural structures and views? Select all that apply.

A. B and D only

B. Views design structures whereas architects document the structures.

C. A structure is the set of elements and their relations as they exist in software or hardware.

D. A view is a representation of a structure

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C, D: A structure is the set of elements and their relations as they exist in software or hardware.; A view is a representation of a structure

An architectural structure consists of elements and their relations in the modeled software or hardware. A view represents selected aspects of a structure for a particular purpose. Both C and D therefore express compatible definitions. Views are representations, not agents that design structures, so B is false. The aggregate A includes that false statement and incorrectly excludes the true structure definition.

Why each choice is right or wrong

A - Incorrect: The B-and-D-only combination includes false B and excludes true C.

B - Incorrect: Views represent structures rather than acting as their designers.

C - Correct: A structure is the underlying set of elements and relations.

D - Correct: A view represents selected aspects of a structure.

Primary note proof

Study-note section SE23

> A structure is the elements/relations themselves; a view represents selected aspects. Module structures show code organization, component-and-connector structures runtime interaction, and allocation structures map software to hardware, teams or environment. Processes/threads belong in concurrency views. Evaluate architectures against quality attributes and scenarios; processor allocation, decomposition and style selection can all be architectural concerns.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 121 | lines 6-15

> 6.1.4. Software architecture
> As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept

of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
In this section, we describe a few sample architectures that can be used for different systems. This
is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
representative examples and refer the reader to the literature for more details.

Software Engineering module | PDF page 130 | lines 3-11

6.3. DOCUMENTING SYSTEM DESIGN
System design is documented in the System Design Document (SDD). It describes design goals set by the project, the subsystem decomposition (with UML class diagrams), the hardware/software mapping (with UML deployment diagrams), the data management, the access control, control flow mechanisms, and boundary conditions. The SDD is used to define interfaces between teams of developers and as reference when architecture-level decisions need to be revisited. The audience for the SDD includes the project management, the system architects (i.e., the developers who participate in the system design), and the developers who design and implement each subsystem. The following is an example template for a SDD:

Question sources

Mock exam 4 · Question 45

Question 21 | Q00244

Which testing level isolates one function or class, using stubs or mocks for called dependencies, so its internal branches can be tested directly?

A. System testing

B. Integration testing

C. System and integration testing together

D. Unit testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer D: Unit testing

Unit testing isolates a function, class or small component and may replace dependencies with stubs or mocks. That matches the stated test scope. White-box cases can then target its internal branches, but white-box is a technique rather than a testing level. Integration and system tests cover combined interactions or the full system, so neither matches the isolation specified here.

Why each choice is right or wrong

A - Incorrect: System testing evaluates the complete integrated system.

B - Incorrect: Integration testing targets interactions among combined parts.

C - Incorrect: Combining system and integration levels does not give isolated unit scope.

D - Correct: Unit testing isolates the function or class with replaceable dependencies.

Primary note proof

Study-note section SE35

> Unit testing isolates a function, class or small component, often replacing dependencies with stubs/mocks; a driver invokes the unit and a stub substitutes a called dependency. Integration testing exercises interfaces and interactions among combined parts. System testing checks the complete integrated system against functional and nonfunctional requirements in a representative environment.

Study-note section SE36

> Functional tests check required behavior against the SRS and acceptance criteria. Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits. Usability, security, compatibility and reliability tests target their respective qualities. “Normal operating conditions” alone does not distinguish functional from performance testing.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 24-29

> Path testing
> Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path testing is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.
> The starting point for path testing is the flow graph. A flow graph consists of nodes representing

Software Engineering module | PDF page 148 | lines 19-26

> Unit testing focuses on the building blocks of the software system, that is, objects and subsystems.
> There are three motivations behind focusing on components. First, unit testing reduces the complexity of the overall test activities, allowing us to focus on smaller units of the system. Second, unit testing makes it easier to pinpoint and correct faults, given that few components are involved in the test. Third, unit testing allows parallelism in the testing activities; that is, each component can be tested independently of one another.
> The specific candidates for unit testing are chosen from the object model and the system decomposition of the system. In principle, all the objects developed during the development

Question sources

Mock exam 4 · Question 46

Question 22 | Q00245

Which offered UML diagram models external behavioral goals and actor interactions at a high level, rather than detailed message timing or state evolution?

A. Component diagram

B. Use case diagram

C. Deployment diagram

D. Class Diagram

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: Use case diagram

A use-case diagram presents external roles, their goals and relationships within a chosen system boundary. It therefore matches the specified high-level behavioral scope. Component, deployment and class diagrams describe structural organization, allocation or static types. Sequence, activity and state diagrams would be appropriate for detailed interactions, workflows or state changes, but those finer-grained dynamic questions are explicitly outside this stem.

Why each choice is right or wrong

A - Incorrect: Component diagrams describe components and interfaces.

B - Correct: Use-case diagrams show actor roles and high-level goals.

C - Incorrect: Deployment diagrams allocate artifacts to nodes.

D - Incorrect: Class diagrams show static types and relationships.

Primary note proof

Study-note section SE20

> An activity diagram models workflow: filled circle start, rounded actions, guarded decision diamond, thick fork/join bars and partitions/swimlanes for responsibility. A diamond can also merge alternative flows; a join synchronizes concurrent flows. A state machine models states and transitions triggered by events, often with guards/actions, for one object's lifecycle.

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 101 | lines 5-6

> In UML, the functional model is represented with use case diagrams, the object model with class diagrams, and the dynamic model with statechart and sequence diagrams.

Software Engineering module | PDF page 47 | lines 30-39

> Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
> Use cases describe the behavior of the system as seen from an actor's point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
> use case to access the system functionality. The use case can then initiate other use cases and gather
> more information from the actors. When actors and use cases exchange information, they are said
> to communicate. We will see later that we represent these exchanges with communication relationships.

Question sources

Mock exam 4 · Question 47

Question 23 | Q00246

Which choice most completely identifies the listed reasons requirements elicitation can be difficult?

A. All choices are correct

B. Problem of scope

C. Problem of understanding

D. Problem of volatility

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: All choices are correct

SE11 identifies scope, understanding and volatility as distinct elicitation difficulties. Scope concerns agreeing what belongs in the system; understanding concerns discovering and interpreting actual needs; volatility concerns needs changing as the project progresses. Since all three listed problems are valid, the inclusive answer is appropriate rather than choosing just one source of difficulty.

Why each choice is right or wrong

A - Correct: All three listed difficulties are explicitly recognized in the complete note.

B - Incorrect: True as one component, but incomplete as the requested full selection. Unclear system boundaries make agreeing on included and excluded requirements difficult.

C - Incorrect: True as one component, but incomplete as the requested full selection. Different interpretations and tacit stakeholder needs complicate accurate understanding.

D - Incorrect: True as one component, but incomplete as the requested full selection. Changing needs make elicitation an ongoing activity rather than a one-time capture.

Primary note proof

Study-note section SE11

> Correct requirements reflect actual stakeholder needs. Complete requirements address the agreed scope, normal and exceptional behavior. Consistency excludes contradictions. Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable within constraints. Verifiability means objective evidence can check fulfillment. Replace “fast for most users” with a defined metric, workload and threshold.

Study-note section SE11

> Trace each requirement backward to its source/rationale and forward to design, code and tests. Record dependencies among requirements. A traceability matrix links these artifacts and helps impact analysis; it is not just a cost/schedule ledger. Stakeholders must negotiate conflicting requirements rather than independently approving incompatible demands. Scope, understanding and volatility are distinct elicitation difficulties.

Study-note section SE-S01

> Elicitation difficulties have several distinct sources. Scope problems concern agreeing on system boundaries and the appropriate level of detail. Understanding problems arise when stakeholders and analysts have different domain knowledge, omit tacit needs or communicate conflicting interpretations. Volatility means that requirements change as circumstances and understanding develop.

Study-note section SE-S01

> These difficulties call for repeated communication, explicit boundaries, negotiated meanings and controlled revision. They can occur together; resolving one does not automatically remove the others.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 26 | lines 6-9

> 2. Requirement’s elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

Software Engineering module | PDF page 68 | lines 18-26

system specification. Requirement validation involves checking if the specification is correct, complete, consistent, unambiguous, and realistic. A specification is correct if it represents the client's view of the system (i.e., everything in the requirements model accurately represents an aspect of the system). It is complete if all possible scenarios through the system are described, including exceptional behavior (i.e., all aspects of the system are represented in the requirements
model). The system specification is consistent if it does not contradict itself. The system specification is unambiguous if exactly one system is defined (i.e., it is not possible to interpret the specification two or more different ways). Finally, it is realistic if the system can be implemented within constraints. These properties are illustrated with UML instance diagrams in

Software Engineering module | PDF page 41 | lines 16-17

✓ Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.

Question sources

Mock exam 4 · Question 48

Question 24 | Q00247

Which development activity primarily recognizes and clarifies the problem, required behavior and related constraints before choosing the solution architecture or algorithms?

A. Implementation

B. Testing

C. Analysis

D. Design

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Analysis

Analysis clarifies the problem, required behavior and constraints. Design then chooses architecture, interfaces and algorithms for a solution; implementation writes it, and testing evaluates its behavior. The revised scope keeps these responsibilities distinct instead of mixing problem recognition with selecting methods to overcome it. Creativity can occur in several activities, so the activity's purpose, rather than that adjective, determines the answer.

Why each choice is right or wrong

A - Incorrect: Implementation writes the selected solution.

B - Incorrect: Testing evaluates behavior.

C - Correct: Analysis clarifies the problem and constraints.

D - Incorrect: Design selects the solution architecture and mechanisms.

Primary note proof

Study-note section SE02

A common lifecycle names requirements, design, implementation, testing, deployment and maintenance. Analysis clarifies the problem; system design chooses architecture; object/detail design specifies interfaces and algorithms; implementation writes code; testing evaluates behavior; deployment puts it into service. These activities may overlap and repeat.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 100 | lines 11-24

Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users
Figure 4.1 Products of requirements elicitation and analysis (UML activity diagram).
Althoughtheanalysismodelmaynotbeunderstandabletotheusersandtheclient, it helps developers verify the system specification produced during requirements elicitation. There is a natural tendency for users and developers to postpone difficult decisions until later in the project. A decision may be difficult because of lack of domain knowledge, lack of technological knowledge, or simply because of disagreements among users and developers. Postponing decisions enables the
project to move on smoothly and avoid confrontation with reality or peers. Unfortunately, difficult
decisions will eventually need to be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation. Translating a system specification in to a formal or semi-formal model forces developers to identify and resolve difficult issues early in
the development.

Software Engineering module | PDF page 18 | lines 13-20

System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
Software design involves identifying and describing the fundamental software system abstractions
and their relationships. The goal of the design phase is to transform the requirements specified in
the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Question sources

Mock exam 4 · Question 49

Question 25 | Q00248

_____ is a set of notations and conventions used to describe and model an application.

A. All choices are correct

B. Unified approach (UA)

C. Unified modeling language (UML)

D. Jacobson's use case

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Unified modeling language (UML)

UML provides modeling notations and conventions; it is distinct from a development methodology.

Why each choice is right or wrong

A - Incorrect: UA and use cases are not both equivalent to the language. See PDF p. 32.

B - Incorrect: UA specifies steps for development. See PDF p. 32.

C - Correct: The exact defined modeling notation language. See PDF p. 32.

D - Incorrect: A use case is an interaction model inside the approach. See PDF p. 32.

Primary note proof

Study-note section SE17

A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links. UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process. Structural things describe conceptual or physical structural elements.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 32 | lines 22-27

1.3.2. Overview of the unified approach.
The unified modeling language (UML) is a set of notations and conventions used to describe and model an application. But, the UML does not specify a methodology or what steps to follow to develop an application; that would be the task of the UA. Figure 1.2 depicts the essence of the unified approach. The heart of the UA is Jacobson's use case. The use case represents a typical interaction between a user and a computer system to capture the users' goals and needs. The main

Question sources

Mock exam 4 · Question 50

Question 26 | Q00249

Which structural risk can arise when repeated software changes are made without coherent design or refactoring?

A. Reduced user engagement

B. Increased software complexity

C. Increased automation

D. Elimination of maintenance costs

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer B: Increased software complexity

Repeated changes without coherent boundaries can accumulate dependencies and make the evolving system harder to understand. Refactoring and good subsystem design help control that structural complexity while maintenance still requires impact analysis and regression checks. User engagement is a product-use outcome rather than the structural risk asked for. Automation is an execution technique, not itself structural deterioration, and ongoing evolution does not eliminate maintenance costs.

Why each choice is right or wrong

A - Incorrect: Engagement concerns user behavior rather than the internal structural deterioration asked for.

B - Correct: Accumulated poorly coordinated changes can increase structural complexity.

C - Incorrect: Automation does not itself describe deterioration of responsibilities or dependency structure.

D - Incorrect: Changing software still requires maintenance and regression work.

Primary note proof

Study-note section SE01

> Software engineering applies systematic methods to development, operation and maintenance. A software product includes executable code and relevant requirements, designs, test assets, configuration and user documentation. A process defines activities, ordering, roles, products and entry/exit conditions. Its four broad activities are specification, design/implementation, validation and evolution.

Study-note section SE40

Corrective maintenance fixes faults; adaptive maintenance responds to environment changes; perfective maintenance improves useful capability/performance; preventive maintenance reduces future risk, such as restructuring fragile code. Motive determines category; refactoring can support preventive or perfective work. Maintenance often lasts years and includes understanding legacy code and regression effort, so it may dominate lifecycle cost.

Study-note section SE22

Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module's internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Study-note section SE01

poor object design can still be complex.

Study-note section SE43

Refactor to preserve behavior while improving structure; optimize only against measured needs; record rationale and update traceability and regression evidence after change.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 41 | lines 16-17

✓ Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.

Software Engineering module | PDF page 19 | lines 24-29

5. Operation and maintenance: Normally (although not necessarily), this is the longest life cycle phase. Maintenance of a typical software product requires much more than the effort necessary to
develop the product itself. The system is installed and put into practical use. Maintenance involves
correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system ‘s services as new requirements are discovered.

Software Engineering module | PDF page 120 | lines 6-12

6.1.3. Coupling and coherence
Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Question sources

Mock exam 4 · Question 51

Question 27 | Q00250

Which of the following best describes the difference between CPM and PERT?

A. CPM focuses on minimizing cost, while PERT focuses on minimizing time.

B. CPM is used for projects with uncertain activity durations, while PERT is used for projects with known activity durations.

C. PERT is used for projects with uncertain activity durations, while CPM is used for projects with known activity durations.

D. PERT is deterministic, whereas CPM is probabilistic.

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer C: PERT models uncertain durations; traditional CPM uses point durations.

SE30 gives traditional CPM point estimates and PERT optimistic, most-likely and pessimistic duration estimates. PERT therefore represents duration uncertainty, whereas the traditional CPM calculation uses known or fixed activity durations. The cost-versus-time contrast is not the defining distinction here, and the other two choices reverse the described treatment of uncertainty.

Why each choice is right or wrong

A - Incorrect: The note's defining contrast concerns duration uncertainty rather than assigning cost exclusively to CPM.

B - Incorrect: This reverses the models: PERT explicitly represents uncertain activity durations.

C - Correct: PERT uses uncertain three-point estimates, while traditional CPM uses fixed point durations.

D - Incorrect: The probabilistic/deterministic characterization is reversed relative to the note.

Primary note proof

Study-note section SE30

> Traditional CPM uses point durations; PERT models uncertainty with optimistic a, most-likely m and pessimistic b. The classic estimate is TE=(a+4m+b)/6 and variance=((b-a)/6)^2. For 3,6,15 days, TE=42/6=7 days and variance=4 days^2. Do not replace the weighted estimate by a simple average. These formulas are approximations, not guaranteed actual completion times.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 3-12

4.6. Scheduling, Organization and Team Structures
Project-task scheduling is an important project planning activity. It involves deciding which tasks would
be taken up when. In order to schedule the project activities, a software project manager needs to do
the following:
• Identify all the tasks needed to complete the project and Break down large tasks into small activities.
• Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
• Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Mock exam 4 · Question 52

Question 28 | Q00251

A key benefit of using a project management tool in software development is:

A. Automatic bug detection and fixing.

B. Replacing the need for human project managers.

C. Improved communication and collaboration within the team.

D. Increased coding efficiency for developers.

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer C: Improved team communication and collaboration.

SE28 says planning tools coordinate ownership, dependencies and progress while leaving judgment to people. That coordination supports communication and collaboration. The tool does not automatically detect and repair code faults, replace project managers or directly guarantee faster coding. Those benefits would require other capabilities and evidence beyond project-management tooling itself.

Why each choice is right or wrong

A - Incorrect: Project-management tools do not automatically detect and repair software defects.

B - Incorrect: They support planning and coordination while human managers still supply judgment.

C - Correct: Shared ownership, dependency and progress information helps the team communicate and collaborate.

D - Incorrect: A direct increase in coding efficiency is not the general coordination benefit established here.

Primary note proof

Study-note section SE28

Planning is continuous: identify scope/deliverables, decompose work, estimate effort/duration/cost, assign resources, sequence dependencies and revise as evidence changes. A WBS is hierarchical and covers project scope without adding unrelated work. Tools help coordinate ownership, dependencies and progress; they do not replace judgment or automatically fix bugs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 90 | lines 7-24

4.2. Project Planning
• Probably the most time-consuming project management activity.
• Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
• Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
The project plan sets out:
• The resources available to the project;
• The work breakdown;
• A schedule for the work.
Project Plan Structure
• Introduction.
• Project organisation.
• Risk analysis.
• Hardware and software resource requirements.
• Work breakdown.
• Project schedule.
• Monitoring and reporting mechanisms.

Software Engineering module | PDF page 81 | lines 11-21

3.4.2. Negotiating specifications with clients: Joint Application Design
Joint Application Design (JAD) is effectiveness lies in that the requirements elicitation work is done in one single workshop session in which all stakeholders participate. Users, clients, developers, and a trained session leader sit together in one room to present their viewpoint, listen
to other viewpoints, negotiate, and agree on a mutually acceptable solution. The outcome of the workshop, the final JAD document, is a complete system specification document that includes definitions of data elements, work flows, and interface screens. Because the final document is jointly developed by the stakeholders (that is, the participants who not only have an interest in the
success of the project, but also can make substantial decisions) the final JAD document represents

an agreement between users, clients, and developers, and thus minimizes requirements changes later in the development process.

Question sources

Mock exam 4 · Question 53

Question 29 | Q00252

A software development team is setting up a configuration management process. They need to identify the configuration items (CIs) for their project. Which of the following would not typically be considered a configuration item?

A. Internet browsers used by the development team

B. Source code files

C. User manuals

D. Test plans

Hint

Separate artifact history and controlled changes from scheduling, collaboration benefits and automated execution.

Answer A: Ordinary browsers used by the team are usually tools, not product configuration items.

Controlled configuration items include source files, requirements, manuals and test plans. A team's ordinary browser is normally just a tool. A particular browser version can become an environment CI if the project deliberately controls it for reproducible testing, but the word typically makes A the appropriate exception in the stated scenario.

Why each choice is right or wrong

A - Correct: An ordinary team browser is usually a tool; a deliberately controlled test-browser version would be an exception.

B - Incorrect: Source code files are standard controlled product configuration items.

C - Incorrect: User manuals can be versioned and baselined alongside the software product.

D - Incorrect: Test plans are controlled project artifacts whose versions must match the tested configuration.

Primary note proof

Study-note section SE33

Configuration management identifies controlled items, versions/baselines, change procedures, status and audits. Items may include source code, requirements, manuals, test plans, environments and build configurations. A team’s ordinary browser is usually a tool rather than product CI, but a controlled test-browser version may be an environment CI.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 5-10

> Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
> faults if change is controlled.

Software Engineering module | PDF page 89 | lines 17-24

> Assigning a role to a participant usually translates into assigning the responsibility of a specific work product. A work product is an artifact that is produced during the development, such as an internal document for other project participants or a deliverable to the customer.
> An analyst is responsible for a section of the RAD describing the functionality of the system. A system architect is responsible for the SDD describing the system's high-level architecture. A project manager is responsible for the SPMP describing the development processes. Finally, the system and its accompanying documentation also constitute a set of work products that are delivered to the client.

Question sources

Mock exam 4 · Question 54

Question 30 | Q00253

Which of the following is NOT a characteristic of a well-defined Work Breakdown Structure (WBS)?

A. It can be used to assign ownership and track the progress of project tasks.

B. It clearly defines deliverables for each level of the breakdown.

C. It includes activities that are outside the project scope.

D. It is organized in a hierarchical manner, with the overall project at the top and smaller tasks at the bottom.

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer C: It includes activities that are outside the project scope.

Planning breaks down the work needed to complete this project. Work outside project scope is therefore not part of its WBS; hierarchy and assignable tasks are consistent with decomposition.

Why each choice is right or wrong

A - Incorrect: Task assignment and tracking serve project management. See PDF p. 90, 95.

B - Incorrect: Deliverables constrain and clarify project work. See PDF p. 90, 95.

C - Correct: Outside-scope activities do not belong to the work needed to complete the defined project. See PDF p. 90, 95.

D - Incorrect: Breaking large tasks into smaller activities supports the hierarchy described. See PDF p. 90, 95.

Primary note proof

Study-note section SE28

Planning is continuous: identify scope/deliverables, decompose work, estimate effort/duration/cost, assign resources, sequence dependencies and revise as evidence changes. A WBS is hierarchical and covers project scope without adding unrelated work. Tools help coordinate ownership, dependencies and progress; they do not replace judgment or automatically fix bugs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 90 | lines 7-24

4.2. Project Planning
• Probably the most time-consuming project management activity.
• Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
• Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
The project plan sets out:
• The resources available to the project;
• The work breakdown;
• A schedule for the work.
Project Plan Structure
• Introduction.
• Project organisation.
• Risk analysis.
• Hardware and software resource requirements.
• Work breakdown.
• Project schedule.
• Monitoring and reporting mechanisms.

Software Engineering module | PDF page 95 | lines 3-12

4.6. Scheduling, Organization and Team Structures
Project-task scheduling is an important project planning activity. It involves deciding which tasks would
be taken up when. In order to schedule the project activities, a software project manager needs to do

the following:
• Identify all the tasks needed to complete the project and Break down large tasks into small activities.
• Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
• Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Mock exam 4 · Question 55

Question 31 | Q00254

A project team is working on a software project with multiple developers. They decided to implement a version control system (VCS). Which of the following scenarios best describes the use case for a distributed VCS like Git?

A. The team prefers to avoid merge conflicts by locking files so only one developer can work on a file at a time.

B. The team is large, geographically distributed, and each developer needs to work on features independently and merge changes later.

C. The team is small, and all developers work in the same physical location on a single branch.

D. The project requires strict control over who can make changes, and all changes must be approved by a central authority before being committed.

Hint

Separate artifact history and controlled changes from scheduling, collaboration benefits and automated execution.

Answer B: Independent distributed development followed by merging.

SE33 says a distributed Git clone contains local history and allows independent branches that can later be merged. B matches that working arrangement directly. Git does not require everyone to lock files, use one local branch or obtain central permission before every local commit. Additional review policies can be imposed, but they do not define distributed version control.

Why each choice is right or wrong

A - Incorrect: Exclusive file locking is not the defining distributed Git collaboration model.

B - Correct: Local histories and independent branches support geographically distributed work followed by merging.

C - Incorrect: A colocated team can use Git, but this scenario does not express its independent distributed-work benefit.

D - Incorrect: Central approval policies can exist, but they do not define or require distributed local commits.

Primary note proof

Study-note section SE33

> Change management handles request, impact assessment, decision/prioritization, implementation and validation; configuration management keeps all affected versions consistent. Distributed Git gives each clone local history and supports independent branches followed by merging; it does not eliminate merge conflicts or approvals. Plan test configuration management early and apply it throughout testing, not at just one exclusive phase.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 5-10

> Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
> faults if change is controlled.

Question sources

Mock exam 4 · Question 56

Question 32 | Q00255

The given statement describes which type of requirement? "A heat sensor detects an intrusion and alerts the system."

A. Known Requirement

B. None of the given options

C. Functional

D. Non-Functional

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Functional

Detection followed by an alert specifies an externally observable service, not a timing or quality constraint.

Why each choice is right or wrong

A - Incorrect: Known requirement is not the module's classification for this behavior. See PDF p. 65, 66.

B - Incorrect: A functional option is present. See PDF p. 65, 66.

C - Correct: Describes the system's required interaction with the intrusion sensor and alert recipient. See PDF p. 65, 66.

D - Incorrect: No performance or other quality bound is stated. See PDF p. 65, 66.

Primary note proof

Study-note section SE09

> A functional requirement states an observable service or behavior, including reactions to inputs, errors and forbidden actions. “When the sensor detects intrusion, send an alert” is functional. “Only a manager may approve refunds” also specifies permitted/prohibited behavior. A negative statement is not automatically nonfunctional.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 65 | lines 19-23

> 3.2.1. Functional Requirements
> Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. For example, the following is an example of functional requirements for SatWatch, a watch that resets itself without user intervention:

Software Engineering module | PDF page 66 | lines 14-18

> 3.2.2. Non-Functional and Pseudo-Requirements
> Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Nonfunctional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands)
> or accuracy (i.e., how precise are the system’s numerical answers). The following are the

Question sources

Mock exam 4 · Question 57

Question 33 | Q00256

In the ordinary software lifecycle terminology used here, which document organizes the agreed user and detailed system requirements?

A. Software Design Description

B. Software Requirement Specification

C. Capability Development Document

D. Domain-Driven Design

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Software Requirement Specification

The SRS records the agreed software services and constraints, including user-level and detailed system requirements. A Software Design Description explains how the solution is organized rather than being the requirements agreement. A Capability Development Document is a specialized defense capability-requirements artifact. Domain-driven design is an approach to organizing software models around business concepts. Thus the ordinary software requirements document requested is the SRS.

Why each choice is right or wrong

A - Incorrect: A design description records the solution structure rather than the agreed requirements specification.

B - Correct: The SRS organizes agreed user and system services and constraints.

C - Incorrect: A Capability Development Document belongs to a specialized defense capability-acquisition context.

D - Incorrect: Domain-driven design is a modeling/design approach, not the name of the requirements document.

Primary note proof

Study-note section SE08

> User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements.

Study-note section SE21

> System design turns the analysis model into architecture: design goals, subsystem responsibilities, dependencies, hardware mapping, data management, access control, control flow and startup/shutdown/error policies.

Study-note section SE-S06

> A Software Requirements Specification records agreed software services and constraints. A Capability Development Document is a defense capability-requirements artifact describing needed capability characteristics and performance parameters; its specialized acquisition context differs from the ordinary software SRS. Domain-driven design is an approach that organizes software models around the business domain, not the name of an agreed requirements specification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 26 | lines 10-17

3. Requirement’s specification: Requirements specification is the activity of translating the information gathered during the analysis activity in to a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are
abstract statements of the system requirements for the customer and end-user of the system; system
requirements are a more detailed description of the functionality to be provided.
4. Requirement’s validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered.
It must then be modified to correct these problems.

Software Engineering module | PDF page 18 | lines 13-20

System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
Software design involves identifying and describing the fundamental software system abstractions
and their relationships. The goal of the design phase is to transform the requirements specified in
the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Question sources

Mock exam 4 · Question 58

Question 34 | Q00257

Which listed statements describe supported advantages or uses of incremental delivery? Select all that apply. Interpret easier testing/debugging as a potential benefit of smaller changes, not a guarantee.

A. Easier to test and debug

B. Easier to test and debug & It is used when there is a need to get a product to the market early

C. Customer can respond to each increment

D. It is used when there is a need to get a product to the market early

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A, B, C, D: Easier to test and debug; Easier to test and debug & It is used when there is a need to get a product to the market early; Customer can respond to each increment; It is used when there is a need to get a product to the market early

Incremental delivery adds usable capability in stages. Smaller changes can be easier to test and debug, users can provide feedback after increments, and prioritized partial capability can reach the market early. Thus A, C and D each describe a supported benefit, and B is also true because it combines two true points. These benefits still require architecture, integration and testing effort.

Why each choice is right or wrong

A - Correct: Smaller increments can be easier to test and debug.

B - Correct: Both the smaller-change and early-delivery points are supported.

C - Correct: Users can respond to each usable increment.

D - Correct: Early delivery of prioritized capability is a supported use.

Primary note proof

Study-note section SE04

> Choose incremental delivery when prioritized partial capability must reach users early. Smaller changes can be easier to debug and permit feedback, though architecture and integration still require planning. Iteration need not rebuild the whole system every time, and increments need not be perfectly independent modules. Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 22 | lines 24-29

> Advantages of Incremental Model:
> • Generates working software quickly and early during the software life cycle. This model is more flexible - less costly to change scope and requirements. It is easier to test and debug during a smaller iteration.
> • In this model customer can respond to each built. Lowers initial delivery cost.
> • Easier to manage risk because risky pieces are identified and handled during its iteration.

Question sources

Mock exam 4 · Question 59

Question 35 | Q00258

Use a process diagram whose named stages are requirements gathering, quick design, prototype construction, prototype refinement and engineering the production product. Which offered activity is not a separately named stage, although it can occur within prototype construction?

A. Coding

B. Prototype Refinement

C. Engineer Product

D. Quick Design

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Coding

Coding can be part of building an executable prototype even if the process diagram labels that work construction. It is therefore the offered activity that is not a separately named stage in the stated convention. Quick design, refinement and production engineering are named stages there. This answer concerns the diagram's labels and does not claim that prototyping excludes coding or always follows one universal phase taxonomy.

Why each choice is right or wrong

A - Correct: Coding can occur inside construction without a separate stage label.

B - Incorrect: Prototype refinement is a named stage.

C - Incorrect: Engineering the production product is a named stage.

D - Incorrect: Quick design is a named stage.

Primary note proof

Study-note section SE07

> A prototype explores requirements, usability or technical feasibility. A throwaway prototype is discarded after learning; an evolutionary prototype is deliberately refined toward production. A quick prototype may omit robustness, security or architecture, so do not deploy it unchanged without evaluation. Coding may occur during prototyping even if a diagram labels that activity construction rather than coding.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 82 | lines 23-29

> • Prototype test. During this type of test, one or more users are presented with a piece of software that practically implements the system. A vertical prototype implements a complete use case through the system, and a horizontal prototype presents an interface for most use cases (without providing much or any functionality). The advantages of prototype

tests are that they provide a realistic view of the system to the user and that prototypes can be instrumented to collect detailed data. The disadvantages of prototypes are that they are expensive to build and to modify.

Question sources

Mock exam 4 · Question 60

Question 36 | Q00259

What is the major drawback of using the RAD Model?

A. Increases reusability of components, Highly specialized & skilled developers/designers are required

B. Increases reusability of components

C. Highly specialized & skilled developers/designers are required

D. Encourages customer/client feedback

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: The need for highly skilled developers and designers.

SE07 describes RAD as rapid construction using timeboxes, user participation and often reusable components, but requiring skilled designers and developers. That staffing prerequisite is the listed drawback. Reuse and customer feedback are useful features. A combines the genuine prerequisite with a benefit, so C states the drawback accurately without misclassifying reuse.

Why each choice is right or wrong

A - Incorrect: It bundles a genuine staffing drawback with component reuse, which is generally a benefit.

B - Incorrect: Component reuse helps rapid construction and is not the drawback asked for.

C - Correct: RAD depends on skilled designers and developers, which can constrain staffing and feasibility.

D - Incorrect: Customer feedback supports learning and fit rather than being the stated RAD disadvantage.

Primary note proof

Study-note section SE07

RAD uses rapid construction, timeboxing, user participation and often reusable components; it requires available users and skilled designers/developers. Reuse-oriented development searches components, evaluates fit, may adjust requirements, designs around interfaces and builds

missing parts. Product-line assets may include requirements, architecture, components and tests; all are potentially reusable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 24 | lines 20-29

Advantages:
1. Reduced cost and risks
2. Faster delivery of the software (Accelerated development)
Disadvantages:
1. Compromising requirements
2. Some control over the system evolution is lost as new versions of the reusable components are not under the control of the organization using them.
3. Increased maintenance costs
4. Not-invented-here syndrome
5. Lack of tool support

Question sources

Mock exam 4 · Question 61

Question 37 | Q00260

Which requirements are the foundation from which quality is measured?

A. None of the mentioned

B. Hardware

C. Software

D. Programmers

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer C: Software

Conformance is evaluated against software behavior specifications, making software requirements the foundation for the quality comparison here.

Why each choice is right or wrong

A - Incorrect: A software requirements option is present. See PDF p. 140, 26.

B - Incorrect: Hardware alone does not specify expected software behavior. See PDF p. 140, 26.

C - Correct: Defines the behavior against which reliability and failure are judged. See PDF p. 140, 26.

D - Incorrect: Programmer preferences are not the behavior specification. See PDF p. 140, 26.

Primary note proof

Study-note section SE34

Quality assurance establishes and checks processes/standards; quality control evaluates work products. An SQA plan identifies standards, responsibilities, evaluations, reviews, audits and assurance records. Quality of conformance means following the design/specification; a product can conform to a poor specification. Verification asks whether specified obligations are implemented correctly; validation asks whether the product meets intended stakeholder needs. Both can use reviews, analysis or execution, so “human-only versus computer-only” is false.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 140 | lines 11-18

Reliability is a measure of success with which the observed behavior of a system conforms to the specification of its behavior. Software reliability is the probability that a software system will not cause the failure of the system for a specified time under specified conditions. Failure is any deviation of the observed behavior from the specified behavior. An error means the system is in a state such that further processing by the system will lead to a failure, which causes the system to
deviate from its intended behavior. A fault, also called defect or bug, is the mechanical or algorithmic cause of an error. The goal of testing is to maximize the number of discovered faults, which then allows developers to correct them and increase the reliability of the system.

Software Engineering module | PDF page 26 | lines 10-17

3. Requirement’s specification: Requirements specification is the activity of translating the information gathered during the analysis activity in to a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are
abstract statements of the system requirements for the customer and end-user of the system; system
requirements are a more detailed description of the functionality to be provided.
4. Requirement’s validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered.
It must then be modified to correct these problems.

Question sources

Mock exam 4 · Question 62

Question 38 | Q00261

Which of the following is not a SQA plan for a project?

A. Documents to be produced by the SQA group

B. Evaluations to be performed

C. Audits and reviews to be performed

D. Amount of technical work

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer D: Amount of general technical development work.

SE34 identifies assurance responsibilities, evaluations, reviews, audits and assurance records as SQA-plan contents. The documents produced by SQA are assurance records; evaluations, audits and reviews are assurance activities. The amount of general technical development work belongs primarily to project scope and effort planning rather than defining the SQA activities listed here.

Why each choice is right or wrong

A - Incorrect: Assurance documents and records are legitimate SQA-plan deliverables.

B - Incorrect: Planned evaluations are explicitly part of an SQA plan.

C - Incorrect: Reviews and audits are explicitly listed assurance-plan activities.

D - Correct: General technical-work quantity is a project scope/effort concern rather than the listed assurance-plan content.

Primary note proof

Study-note section SE34

> Quality assurance establishes and checks processes/standards; quality control evaluates work products. An SQA plan identifies standards, responsibilities, evaluations, reviews, audits and assurance records. Quality of conformance means following the design/specification; a product can conform to a poor specification. Verification asks whether specified obligations are implemented correctly; validation asks whether the product meets intended stakeholder needs. Both can use reviews, analysis or execution, so “human-only versus computer-only” is false.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 99 | lines 14-19

> 4.8. Quality Assurance Monitoring Plans
> Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product
> quality, thereby gaining insight and confidence that product quality is meeting its goals. Of

course,
if the data provided through quality assurance identify problems, it is management's responsibility
to address the problems and apply the necessary resources to resolve quality issues.

Question sources

Mock exam 4 · Question 63

Question 39 | Q00262

Which of the below is not a part of the Test Plan?

A. Schedule

B. Incident reports

C. Entry and exit criteria

D. Risk

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: Incident reports are execution outputs, rather than the test plan itself.

SE39 places objectives, scope, resources, schedule, risks and entry/exit criteria in test planning. Incident reports document observed problems during execution and are normally outputs of that work. They may be referenced by later plans, but that does not make actual incident reports an intrinsic part of the prospective test plan.

Why each choice is right or wrong

A - Incorrect: The test schedule is planned work and belongs in a test plan.

B - Correct: Incident reports normally record execution findings rather than describing the planned testing work.

C - Incorrect: Entry and exit criteria define when testing can start or finish and belong in planning.

D - Incorrect: Testing risks influence scope and priorities and are legitimate test-plan content.

Primary note proof

Study-note section SE39

A test plan describes intended work; incident/defect reports are normally execution outputs. A defect report should identify build/environment, steps, expected versus actual behavior, evidence and severity. Anyone finding a fault should record it; testers often coordinate reporting, but it is not their exclusive permission. Configuration control makes the tested version reproducible.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 90 | lines 7-24

4.2. Project Planning
• Probably the most time-consuming project management activity.
• Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
• Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
The project plan sets out:
• The resources available to the project;
• The work breakdown;
• A schedule for the work.
Project Plan Structure
• Introduction.
• Project organisation.
• Risk analysis.
• Hardware and software resource requirements.
• Work breakdown.
• Project schedule.
• Monitoring and reporting mechanisms.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Software Engineering module | PDF page 19 | lines 20-23

System testing is normally carried out in a planned manner according to the system test plan document. The system test plan identifies all testing-related activities that must be performed, specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.

Question sources

Mock exam 4 · Question 64

Question 40 | Q00263

Exploratory testing is a -----?

A. Experience-based Test Design Technique

B. Black Box Test Design Technique

C. White Box Test Design Technique

D. Grey Box Test Design Technique

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer A: Experience-based testing.

SE36 explicitly classifies exploratory testing as experience based because learning, test design and execution occur together. It may use knowledge of specifications or internals, but black-box, white-box and grey-box describe different bases of test derivation. None of them replaces the defining experience-based classification asked for in this question.

Why each choice is right or wrong

A - Correct: Exploratory testing uses experience while learning, designing and executing tests together.

B - Incorrect: It may use external behavior, but black-box alone is not its defining classification.

C - Incorrect: Exploratory testing does not require test cases to be derived from internal code structure.

D - Incorrect: Partial internal knowledge can help exploration, but it does not define the technique.

Primary note proof

Study-note section SE36

> Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair. Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 6-10

> Equivalence testing is a black box testing technique that minimizes the number of test cases. The possible inputs are partitioned into equivalence classes, and a test case is selected for each class. The assumption of equivalence testing is that systems usually behave in similar ways for all members of a class. To test the behavior associated with an equivalence class, we only need to test
> one member of the class. Equivalence testing consists of two steps: identification of the

Software Engineering module | PDF page 149 | lines 24-29

> Path testing
> Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path testing is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.
> The starting point for path testing is the flow graph. A flow graph consists of nodes representing

Question sources

Mock exam 4 · Question 65

Question 41 | Q00264

Non-conformance to software requirements is known as

A. Software reliability

B. Software availability

C. None of the mentioned

D. Software failure

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer D: Software failure

Failure is the deviation of observed behavior from its specified behavior; reliability measures conformance rather than the non-conformance event.

Why each choice is right or wrong

A - Incorrect: Probability or measure of success, not the deviation itself. See PDF p. 140.

B - Incorrect: Availability is not defined as non-conformance. See PDF p. 140.

C - Incorrect: The failure option supplies the defined term. See PDF p. 140.

D - Correct: Matches deviation from specified behavior. See PDF p. 140.

Primary note proof

Study-note section SE34

A fault/defect is an underlying problem; an error may be an incorrect internal state; a failure is an externally observed departure from required service. Reliability is a product quality attribute; low cost or short development time are project outcomes. Tests increase confidence and reveal defects but cannot generally guarantee correctness over every possible input.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 140 | lines 11-18

Reliability is a measure of success with which the observed behavior of a system conforms to the specification of its behavior. Software reliability is the probability that a software system will not cause the failure of the system for a specified time under specified conditions. Failure is any deviation of the observed behavior from the specified behavior. An error means the system is in a state such that further processing by the system will lead to a failure, which causes the system to
deviate from its intended behavior. A fault, also called defect or bug, is the mechanical or algorithmic cause of an error. The goal of testing is to maximize the number of discovered faults, which then allows developers to correct them and increase the reliability of the system.

Question sources

Mock exam 4 · Question 66

Question 42 | Q00308

When comparing internal subsystem-grouping criteria with an external user perspective, which option names the external perspective rather than an internal structural or responsibility criterion?

A. Coupling

B. System's functionality

C. Cohesion

D. Users thinking

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: Users thinking

Coupling measures dependencies across subsystem boundaries, cohesion measures relatedness within a subsystem, and functionality helps assign responsibilities. These are internal structural grouping criteria. A user’s perspective describes externally needed tasks and behavior, so it is the different kind of information requested. User workflows can still inform architecture; the distinction does not claim users have no influence on design or that internal boundaries should ignore their needs.

Why each choice is right or wrong

A - Incorrect: Coupling directly measures inter-subsystem dependency.

B - Incorrect: Functionality guides internal responsibility grouping.

C - Incorrect: Cohesion measures how well elements grouped inside a subsystem serve a related purpose.

D - Correct: This names the external user perspective rather than an internal structural measure or responsibility criterion.

Primary note proof

Study-note section SE21

> Group closely related objects and functionality, minimize associations crossing boundaries, and isolate communication infrastructure where useful. User workflows inform decomposition but do not alone dictate the internal structure. Analysis describes domain needs; subsystem decomposition is principally a design result. Architecture documentation supports communication and records major decisions, not every source-level execution step.

Study-note section SE22

> Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module's internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 116 | lines 16-29

> The analysis model describes the system completely from the actors' point of view and serves as the basis of communication between the client and the developers. The analysis model, however, does not contain information about the internal structure of the system, its hardware configuration,
> or, more generally, how the system should be realized. System design is the first step in this direction. System design results in the following products:
> • List of design goals, describing the qualities of the system that developers should optimize
> • Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.
> The design goals are derived from the nonfunctional requirements. Design goals guide the decisions to be made by the developers especially when trade-offs are needed. The subsystem decomposition constitutes the bulk of system design. Developers divide the system into manageable pieces to deal with complexity: Each subsystem is assigned to a team and realized independently. In order for this to be possible, though, developers need to address system-wide

Software Engineering module | PDF page 120 | lines 6-12

> 6.1.3. Coupling and coherence
> Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely

coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Software Engineering module | PDF page 120 | lines 24-27

Coherence is the strength of dependencies within a subsystem. If a subsystem contains many objects that are related to each other and perform similar tasks, its coherence is high. If a subsystem
contains a number of unrelated objects, its coherence is low. A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence.

Question sources

Computer Science model exam · Question 39

University of Gondar model exam · Question 39

Question 43 | Q00309

The functional model is represented graphically with --

A. Use case diagram

B. State transition diagram

C. Class diagram

D. Object diagram

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: Use case diagram

Use case diagrams represent the functional model; different diagrams represent the dynamic and object models.

Why each choice is right or wrong

A - Correct: Explicit graphical form of the functional model. See PDF p. 101.

B - Incorrect: Models dynamic state transitions. See PDF p. 101.

C - Incorrect: Models object structure. See PDF p. 101.

D - Incorrect: Models object instances, not the functional model. See PDF p. 101.

Primary note proof

Study-note section SE13

> A scenario is one concrete execution; a use case abstracts the related successful and alternative scenarios. Analysis integrates functional (use cases/scenarios), structural/object (classes) and dynamic (states/interactions) models. Entity objects represent domain information such as Incident; boundary objects mediate interaction, such as ReportForm; control objects coordinate a use case. A UI form is normally boundary, not entity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 101 | lines 5-6

> In UML, the functional model is represented with use case diagrams, the object model with class diagrams, and the dynamic model with statechart and sequence diagrams.

Question sources

Computer Science model exam · Question 40

University of Gondar model exam · Question 40

Question 44 | Q00310

Under the course’s distinction between problem analysis and solution design, which statement is false?

A. System Analysis Focuses on the user's view of the system

B. Design focuses on the developer point of view.

C. Result of system analysis is a model that includes a subsystem decomposition

D. Requirements elicitation establishes externally visible stakeholder needs and constraints before internal solution design.

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Result of system analysis is a model that includes a subsystem decomposition

Analysis clarifies the domain and externally required behavior; system design chooses the solution architecture and subsystem responsibilities. Attributing subsystem decomposition to the principal analysis result therefore confuses these stages. Requirements elicitation establishes stakeholder needs and constraints before the internal solution is selected. The revised elicitation option avoids the stronger claim that only one user’s viewpoint matters, while retaining the intended external-needs versus internal-design distinction.

Why each choice is right or wrong

A - Incorrect: Analysis clarifies the user/domain view of required behavior.

B - Incorrect: Design develops the solution structure and implementation-facing decisions.

C - Correct: Subsystem decomposition is principally a system-design result, so attributing it to analysis is the false claim.

D - Incorrect: Elicitation establishes externally visible stakeholder needs and constraints before choosing internal design.

Primary note proof

Study-note section SE02

> Analysis clarifies the problem; system design chooses architecture; object/detail design specifies interfaces and algorithms; implementation writes code; testing evaluates behavior; deployment puts it into service.

Study-note section SE21

> User workflows inform decomposition but do not alone dictate the internal structure. Analysis describes domain needs; subsystem decomposition is principally a design result.

Study-note section SE08

> User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 63 | lines 3-8

> Requirements elicitation and analysis focus only on the user's view of the system. For example, the system functionality, the interaction between the user and the system, the errors that the system
> can detect and handle, and the environmental conditions in which the system functions, are part of
> the requirements. The system structure, the implementation technology selected to build the system, the system design, the development methodology, and other aspects not directly visible to
> the user are not part of the requirements.

Software Engineering module | PDF page 116 | lines 16-29

> The analysis model describes the system completely from the actors' point of view and serves as the basis of communication between the client and the developers. The analysis model, however, does not contain information about the internal structure of the system, its hardware configuration,
> or, more generally, how the system should be realized. System design is the first step in this direction. System design results in the following products:
> • List of design goals, describing the qualities of the system that developers should optimize
> • Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.
> The design goals are derived from the nonfunctional requirements. Design goals guide the decisions to be made by the developers especially when trade-offs are needed. The subsystem decomposition constitutes the bulk of system design. Developers divide the system into manageable pieces to deal with complexity: Each subsystem is assigned to a team and realized independently. In order for this to be possible, though, developers need to address system-wide

Question sources

Computer Science model exam · Question 41

University of Gondar model exam · Question 41

Question 45 | Q00311

Which of the following heuristics is not used to identify Entity objects

A. Real-world entities

B. Real-world activities

C. Forms

D. Data sources

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Forms normally identify boundary objects.

SE13 defines entity objects as domain information and gives ReportForm as a boundary mediating interaction. Thus forms normally suggest boundary objects rather than entity objects. Real-world things, domain activities represented as information and relevant stored data can suggest entity candidates; those are heuristics, not mechanical rules guaranteeing a class for every noun or activity.

Why each choice is right or wrong

A - Incorrect: Real-world domain entities are natural candidates for entity objects.

B - Incorrect: A domain activity represented as persistent domain information can suggest an entity candidate.

C - Correct: A user-interface form normally mediates interaction and is modeled as a boundary object.

D - Incorrect: Relevant domain data sources can suggest information-holding entities, subject to analysis rather than automatic conversion.

Primary note proof

Study-note section SE13

> Use nouns as candidate objects, possessive descriptions as candidate attributes, and verbs such as manages/reports-to as candidate associations. These are heuristics, not automatic parsing rules. CRC means Class, Responsibilities, Collaborators. A control per use case or participating actor can be a useful starting point, not a mandatory UML rule or fixed object lifetime.

Study-note section SE13

> A UI form is normally boundary, not entity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 105 | lines 9-15

> 5.2.1. Identifying Entity Objects
> Requirements Elicitation, participating objects are found by examining each use case and identifying candidate objects. Natural language analysis is an intuitive set of heuristics for identifying objects, attributes, and associations from a system specification. Abbott’s heuristics maps parts of speech (e.g., nouns, having verbs, being verbs, adjectives) to model components (e.g., objects, operations, inheritance relationships, classes). Table 4-1 provides examples of such mappings by examining the Report Emergency use case.

Software Engineering module | PDF page 101 | lines 9-15

The analysis object model consists of entity, boundary, and control objects. Entity objects represent the persistent information tracked by the system. Boundary objects represent the interactions between the actors and the system. Control objects represent the tasks that a formed by the user and supported by the system. In the 2B watch example, Year, Month, Day are entity objects; Button Boundary and LCD Display Boundary are boundary objects; Change Date Control is
a control object that represents the activity of changing the date by pressing combinations of buttons.

Question sources

Computer Science model exam · Question 42

University of Gondar model exam · Question 42

Question 46 | Q00312

"Consider a system where, a heat sensor detects an intrusion and alerts the security company." What kind of a requirement the system is providing?

A. Functional Requirement

B. Non-Functional

C. Constraints

D. Known Requirement

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: Functional Requirement

Detecting an intrusion and notifying the security company is a required interaction/service independent of implementation.

Why each choice is right or wrong

A - Correct: Specifies the system's required behavior. See PDF p. 65.

B - Incorrect: No response-time, reliability or quality limit is stated. See PDF p. 65.

C - Incorrect: No implementation or resource restriction is stated. See PDF p. 65.

D - Incorrect: Known Requirement is not the behavioral category defined here. See PDF p. 65.

Primary note proof

Study-note section SE09

Write one clear obligation with an actor/trigger, action and observable result. “The system shall generate a PDF report” is functional. “The system shall not reveal another customer’s balance” can be a security-related functional rule. Security as a broad quality goal and the functions

implementing it must be distinguished. Words such as must, should and not work do not by themselves determine a requirement’s classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 65 | lines 19-23

3.2.1. Functional Requirements
Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. For example, the following is an example of functional requirements for SatWatch, a watch that resets itself without user intervention:

Question sources

Computer Science model exam · Question 43

University of Gondar model exam · Question 43

Question 47 | Q00313

which one of the following is used to Identifying Actors

A. What kind of dependencies are exist among use cases.

B. What information does the actor access?

C. Which user groups execute the system's main functions?

D. What are the tasks that the actor wants the system to perform?

Hint

Match the information to be represented with each model’s purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Which user groups execute the system's main functions?

Identifying user groups that execute the main functions identifies actors; questions about information and tasks identify scenarios.

Why each choice is right or wrong

A - Incorrect: Concerns use-case relationships rather than identifying external entities. See PDF p. 71, 73.

B - Incorrect: Listed as a scenario-identification question. See PDF p. 71, 73.

C - Correct: Exact actor-identification heuristic. See PDF p. 71, 73.

D - Incorrect: Listed as a scenario-identification question. See PDF p. 71, 73.

Primary note proof

Study-note section SE14

An actor is an external role interacting with the system; it can be a person, device or another system. A person may play several roles, and several people may play one role. A primary actor seeks a goal; a supporting actor provides a service. Ask which user groups perform main functions, administer or maintain the system, and which external systems exchange information.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 71 | lines 13-18

When identifying actors, developers can ask the following questions:
Questions for identifying actors
• Which user groups are supported by the system to perform their work?
• Which user groups execute the system's main functions?
• Which user groups perform secondary functions, such as maintenance and administration?
• Will the system interact with any external hardware or software system?

Software Engineering module | PDF page 73 | lines 4-9

Questions for identifying scenarios
• What are the tasks that the actor wants the system to perform?
• What information does the actor access? Who creates that data? Can it be modified or removed?
By whom?
• Which external changes does the actor need to inform the system about? How often? When?
• Which events does the actor need to be informed by the system about? With what latency?

Question sources

Computer Science model exam · Question 44

University of Gondar model exam · Question 44

Question 48 | Q00547

_____is requirement elicitation techniques that is used to gather the requirements using day-to-day working observation and notes made of the actual tasks.

A. Use case

B. Ethnography

C. Scenario

D. Interview

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Ethnography.

The stem describes observing people's actual daily tasks and recording what they do. SE12 names that observation-based elicitation approach ethnography. Interviews ask stakeholders questions, while scenarios and use cases describe executions or goals; those can complement observation but do not name the day-to-day fieldwork technique itself.

Why each choice is right or wrong

A - Incorrect: A use case describes goal-oriented interactions rather than naming observational fieldwork.

B - Correct: Ethnography gathers understanding by observing actual day-to-day work and recording it.

C - Incorrect: A scenario is one concrete execution, not the observation-based elicitation technique itself.

D - Incorrect: An interview asks people questions rather than primarily observing their daily tasks.

Primary note proof

Study-note section SE12

> Interviews ask stakeholders about needs; document analysis examines manuals, contracts and current records; observation/ethnography studies actual daily work. Prototypes reveal tacit expectations. Workshops/JAD bring users, managers and developers together with a facilitator and recorder to build consensus. Use several sources and check assumptions against observed work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 80 | lines 5-11

> Task analysis is based on the assumption that it is inefficient to ask users to describe what they do
> and how they do it. Users usually do not think explicitly about the sequence of tasks that are required to accomplish their work as they have often repeated these tasks many times. Users, when
> ask how they accomplish their work, would describe, at best, how they are supposed to accomplish
> it, which may be far from reality. Consequently, task analysis uses observation as an alternative to
> build an initial task model. This initial task model is then refined by asking the users why they accomplish a task a certain way.

Question sources

Computer Science model exam · 2015 · Question 16

Question 49 | Q00549

Which one of the following is more related to effective management of software complexity?

A. Modularity

B. Reusability

C. Extensibility

D. All

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer A: Modularity

Decomposition into simpler, manageable subsystems controls complexity. Reuse and extensibility can benefit designs, but do not substitute for this direct mechanism.

Why each choice is right or wrong

A - Correct: Divides the solution domain into manageable simpler parts. See PDF p. 118, 32.

B - Incorrect: Reusability is an additional benefit of object-oriented modules. See PDF p. 118, 32.

C - Incorrect: Extensibility concerns future additions rather than directly partitioning complexity. See PDF p. 118, 32.

D - Incorrect: The text directly establishes modularity; it does not identify all three as equivalent complexity mechanisms. See PDF p. 118, 32.

Primary note proof

Study-note section SE21

> System design turns the analysis model into architecture: design goals, subsystem responsibilities, dependencies, hardware mapping, data management, access control, control flow and startup/shutdown/error policies. Detail/object design refines APIs, parameter/return types, exceptions, component choices and algorithms. A subsystem is a manageable logical collection with services and an interface; it need not be a physical replaceable artifact.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 118 | lines 11-16

> 6.1.1. Subsystems and classes
> In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Software Engineering module | PDF page 32 | lines 12-21

A class in an object-oriented system carefully delineates between its interfaces the routines and attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact
on other classes, and so, the impact is minimized. However, the object-oriented approach is not a panacea; nothing is magical here that will promote perfect design or perfect code.
Promotion of reusability
Objects are reusable because they are modeled directly out of a real-world problem domain. Each
object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.

Question sources

Computer Science model exam · 2015 · Question 18

Question 50 | Q00551

Which one of the following testing required knowledge about internal structure, design and implementation of the system?

A. Black box Testing

B. White box testing

C. Alpha testing

D. Beta testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: White box testing

Path testing requires source code and data-structure knowledge; black-box equivalence testing instead groups observable inputs.

Why each choice is right or wrong

A - Incorrect: Does not require internal paths. See PDF p. 149, 149.

B - Correct: Matches the source-knowledge requirement. See PDF p. 149, 149.

C - Incorrect: Alpha denotes development-team system testing, not an internal-knowledge technique. See PDF p. 149, 149.

D - Incorrect: Beta denotes testing by customers, not an internal-knowledge technique. See PDF p. 149, 149.

Primary note proof

Study-note section SE36

Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair. Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 24-29

Path testing
Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path testing is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.
The starting point for path testing is the flow graph. A flow graph consists of nodes representing

Software Engineering module | PDF page 149 | lines 6-10

Equivalence testing is a black box testing technique that minimizes the number of test cases. The possible inputs are partitioned into equivalence classes, and a test case is selected for each class. The assumption of equivalence testing is that systems usually behave in similar ways for all members of a class. To test the behavior associated with an equivalence class, we only need to test
one member of the class. Equivalence testing consists of two steps: identification of the

Question sources

Computer Science model exam · 2015 · Question 20

Question 51 | Q00554

Which team organization shares development decisions among team members rather than centralizing technical coordination in a chief programmer?

A. Functional

B. Project

C. Democratic

D. Chief programmer

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer C: Democratic

Democratic teams share decisions, whereas chief-programmer teams centralize technical coordination. Functional and project organizations classify how people are grouped, not how authority is distributed. Asking about shared decision-making therefore makes democratic the unique answer. It does not require every individual to perform every lifecycle task; a democratic team can still divide work by expertise and responsibilities.

Why each choice is right or wrong

A - Incorrect: Functional organization groups specialists by discipline.

B - Incorrect: Project organization groups people by project rather than defining shared authority.

C - Correct: Democratic teams share decisions.

D - Incorrect: Chief-programmer teams centralize technical coordination.

Primary note proof

Study-note section SE32

> Functional organizations group specialists by discipline; project organizations group people around projects. Functional specialization helps staffing and knowledge continuity but may create handoff barriers. Democratic teams share decisions; chief-programmer teams centralize technical coordination. These are different classification dimensions. Tuckman’s sequence is forming→storming→norming→performing→adjourning: orientation, conflict, norms, effective work and closure.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 20-26

> Functional format vs. project format
> • There are essentially two broad ways in which a software development organization can be structured:
> • Functional format and project format. In the project format, the project development staffs are divided based on the project for which they work (as shown in next fig).
> • In the functional format, the development staffs are divided based on the functional group to which they belong.

Software Engineering module | PDF page 96 | lines 3-11

> Team Structure
> • Team structure addresses the issue of organization of the individual project teams. There are some possible ways in which the individual project teams can be organized.
> • Problems of different complexities and sizes often require different team structures for chief solution.
> • The most common types of team structure are:
> · Chief Programmer Team
> · Democratic Team
> · Mixed Control Team Organization

Question sources

Computer Science model exam · 2015 · Question 23

Question 52 | Q00555

Which capability maturity model level an organization must master the requirements management software quality assurance, and software project planning

A. Repeatable level

B. Initial level

C. Defined level

D. Managed level

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer A: Historical CMM Level 2: Repeatable.

SE32 explicitly assigns requirements management, software project planning, quality assurance and configuration management to historical Software CMM Level 2. Its name is Repeatable, rather than the later CMMI label Managed. The other named choices correspond to levels 1, 3 and 4 in this historical model, so A matches both practices and terminology.

Why each choice is right or wrong

A - Correct: Repeatable is historical Software CMM Level 2, which establishes the listed basic project practices.

B - Incorrect: Initial is historical Level 1, before the repeatable project discipline specified here.

C - Incorrect: Defined is historical Level 3, not the level associated with this basic practice list.

D - Incorrect: Managed is historical Software CMM Level 4; modern CMMI terminology must not be substituted.

Primary note proof

Study-note section SE32

> Historical Software CMM levels are 1 Initial, 2 Repeatable, 3 Defined, 4 Managed, 5 Optimizing. Level2 establishes basic project discipline including requirements management, project planning, quality assurance and configuration management. These historical labels must not be silently replaced by modern CMMI terminology. A maturity model concerns process capability, not a guarantee that every product is defect free.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 91 | lines 37-42

• Software Metrics can be used to indicate the basic attributes of the software. The indicative metrics, not exhaustive, are
• Size - Line of code ,Function point count
• Quality- Reliability, Accuracy, Dependability, and these are measured through some measures like errors, failures and number of runs between failures ,etc.
• Execution Time -process time of execution of a critical process and many more.

Question sources

Computer Science model exam · 2015 · Question 24

Question 53 | Q00558

All are the advantages of functional format structure, except

A. Ease of staffing

B. Production of good quality documents

C. No need job specialization

D. Efficient handling manpower turnover

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer C: No need for job specialization is false of a functional organization.

SE32 defines functional organizations as grouping specialists by discipline. Claiming that specialization is unnecessary contradicts that organizing principle. Ease of staffing and knowledge continuity are benefits of specialization; continuity can help with turnover. Specialist expertise can also support documentation quality, although no organizational form guarantees good documents. C is the clear exception.

Why each choice is right or wrong

A - Incorrect: Grouping specialists supports staffing within established disciplines, so this is a plausible benefit.

B - Incorrect: Specialist expertise can support good documentation; this is not the direct contradiction of functional grouping.

C - Correct: This contradicts the defining use of specialist disciplines in a functional organization.

D - Incorrect: Knowledge continuity within a functional group can help absorb staff turnover.

Primary note proof

Study-note section SE32

Functional organizations group specialists by discipline; project organizations group people around projects. Functional specialization helps staffing and knowledge continuity but may create handoff barriers. Democratic teams share decisions; chief-programmer teams centralize

technical coordination. These are different classification dimensions. Tuckman's sequence is forming→storming→norming→performing→adjourning: orientation, conflict, norms, effective work and closure.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 20-26

Functional format vs. project format
• There are essentially two broad ways in which a software development organization can be structured:
• Functional format and project format. In the project format, the project development staffs are divided based on the project for which they work (as shown in next fig).
• In the functional format, the development staffs are divided based on the functional group to which they belong.

Question sources

Computer Science model exam · 2015 · Question 27

Question 54 | Q00568

Which testing should focus on testing the functionality of objects or methods?

A. Integration testing

B. White box testing

C. Black box testing

D. Unit testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer D: Unit testing

Unit testing isolates the building blocks, including objects; a method is part of such an individual component's behavior.

Why each choice is right or wrong

A - Incorrect: Tests components together. See PDF p. 148, 144.

B - Incorrect: A technique based on internal paths, not the requested component level. See PDF p. 148, 144.

C - Incorrect: A technique based on external behavior, not a component level. See PDF p. 148, 144.

D - Correct: Tests isolated objects or subsystems and their functionality. See PDF p. 148, 144.

Primary note proof

Study-note section SE35

Acceptance testing supports the customer/user's acceptance decision against agreed needs and criteria. Alpha typically happens in a controlled developer setting with selected users; beta uses external users in their environments. None of these levels is universally the longest. Unit is a level, while white-box is a technique; a unit can be tested using black-box cases too.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 148 | lines 19-26

Unit testing focuses on the building blocks of the software system, that is, objects and subsystems.
There are three motivations behind focusing on components. First, unit testing reduces the complexity of the overall test activities, allowing us to focus on smaller units of the system. Second, unit testing makes it easier to pinpoint and correct faults, given that few components are involved in the test. Third, unit testing allows parallelism in the testing activities; that is, each component can be tested independently of one another.
The specific candidates for unit testing are chosen from the object model and the system decomposition of the system. In principle, all the objects developed during the development

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Question sources

Computer Science model exam · 2015 · Question 37

Question 55 | Q00572

Which of the following views represents the interaction of the user with the software but tells nothing about the internal working of the software?

A. Use case diagram

B. Activity diagram

C. Class diagram

D. Sequence diagram

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer A: Use case diagram

A use case describes external behavior from an actor's point of view without prescribing the internal realization.

Why each choice is right or wrong

A - Correct: Directly models actor-visible interaction. See PDF p. 47, 54, 51.

B - Incorrect: Models action/control flow and may include internal work. See PDF p. 47, 54, 51.

C - Incorrect: Describes classes and their structure. See PDF p. 47, 54, 51.

D - Incorrect: Distributes interaction among participating objects, revealing internal collaboration. See PDF p. 47, 54, 51.

Primary note proof

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

> Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
> Use cases describe the behavior of the system as seen from an actor's point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
> use case to access the system functionality. The use case can then initiate other use cases and

gather
more information from the actors. When actors and use cases exchange information, they are said
to communicate. We will see later that we represent these exchanges with communication relationships.

Software Engineering module | PDF page 54 | lines 26-35

both buttons simultaneously. Figure 2.10 depicts a sequence diagram for an actor setting his 2Bwatch one minute ahead.
Each column represents an object that participates in the interaction. The vertical axis represents
time from top to bottom. Messages are shown by arrows. Labels on arrows represent message names and may contain arguments. Activations (i.e., executing methods) are depicted by vertical rectangles. Actors are shown as the leftmost column.
Sequence diagrams can be used to describe either an abstract sequence (i.e., all possible interactions) or concrete sequences (i.e., one possible interaction, as in Figure 2.10). When describing all possible interactions, sequence diagrams also provide notations for conditionals and
iterators.

Software Engineering module | PDF page 51 | lines 14-24

2.3.2. Class Diagrams
Classes and objects
Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behavior of a set of objects. Objects are entities that encapsulate state and behavior. Each object has an identity: It can be referred individually and is distinguishable from other objects.
In UML, classes and objects are depicted by boxes including three compartments. The top compartment displays the name of the class or object. The center compartment displays its attributes, and the bottom compartment displays its operations. The attribute and operation compartment can be omitted for clarity. Object names are underlined to indicate that they are instances. By convention, class names start with an uppercase letter. Objects in object diagrams

Question sources

Computer Science model exam · 2015 · Question 41

Computer Science model exam · 2015 · Question 126

Question 56 | Q00575

Requirements and technical feasibility are unclear. The immediate goal is to build an exploratory version for user feedback and feasibility learning, then discard it rather than deliver it as production software. Which model best names that immediate activity?

A. Incremental model

B. Prototype model

C. Spiral model

D. Waterfall model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Prototype model

A throwaway prototype explores requirements, usability or technical feasibility and is discarded after learning. That exactly matches the immediate purpose specified. Incremental delivery would provide usable production capability; spiral would organize broader commitments around risk-driven cycles and might use a prototype within them. Classical waterfall favors understood requirements and familiar technology. The specific learning-and-discard goal therefore identifies the prototype model.

Why each choice is right or wrong

A - Incorrect: Incremental delivery targets usable capability rather than a discarded exploratory artifact.

B - Correct: A throwaway prototype explores needs and feasibility before being discarded.

C - Incorrect: Spiral may use a prototype, but names the broader risk-driven process.

D - Incorrect: The stated uncertainty lacks classical waterfall's stable, familiar conditions.

Primary note proof

Study-note section SE07

> A prototype explores requirements, usability or technical feasibility. A throwaway prototype is discarded after learning; an evolutionary prototype is deliberately refined toward production. A quick prototype may omit robustness, security or architecture, so do not deploy it unchanged without evaluation. Coding may occur during prototyping even if a diagram labels that activity construction rather than coding.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 26 | lines 6-9

> 2. Requirement's elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

Software Engineering module | PDF page 82 | lines 23-29

> • Prototype test. During this type of test, one or more users are presented with a piece of software that practically implements the system. A vertical prototype implements a complete use case through the system, and a horizontal prototype presents an interface for most use cases (without providing much or any functionality). The advantages of prototype tests are that they provide a realistic view of the system to the user and that prototypes can be instrumented to collect detailed data. The disadvantages of prototypes are that they are expensive to build and to modify.

Software Engineering module | PDF page 20 | lines 18-22

> the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
> In principle, the waterfall model should only be used when the requirements are well understood
> and unlikely to change radically during system development. However, the waterfall model

Question sources

Computer Science model exam · 2015 · Question 44

Question 57 | Q00580

What is the purpose of a sequence diagram?

A. To show the interactions between actors and a system

B. To show the behavior of objects in a system

C. To show the flow of activities in a system

D. To show the order and timing of interactions between objects

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer D: To show the order and timing of interactions between objects

The vertical axis is time and arrows are messages among participating objects; this directly supports order and timing.

Why each choice is right or wrong

A - Incorrect: Actor interaction is part of some sequences but is less precise than object-message order. See PDF p. 54.

B - Incorrect: Broadly describes behavior without identifying temporal interaction. See PDF p. 54.

C - Incorrect: Activity flow is represented by activity diagrams. See PDF p. 54.

D - Correct: Matches time-ordered messages and activations. See PDF p. 54.

Primary note proof

Study-note section SE19

> Sequence diagrams show interactions across lifelines, with order/time generally downward. Messages cross between participants; activation rectangles show execution. A creation message starts a lifeline; an X marks modeled destruction, not a guarantee about when a garbage collector runs. Synchronous calls wait for completion; asynchronous messages permit the sender to proceed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 54 | lines 26-35

> both buttons simultaneously. Figure 2.10 depicts a sequence diagram for an actor setting his 2Bwatch one minute ahead.
> Each column represents an object that participates in the interaction. The vertical axis represents
> time from top to bottom. Messages are shown by arrows. Labels on arrows represent message names and may contain arguments. Activations (i.e., executing methods) are depicted by vertical rectangles. Actors are shown as the leftmost column.
> Sequence diagrams can be used to describe either an abstract sequence (i.e., all possible interactions) or concrete sequences (i.e., one possible interaction, as in Figure 2.10). When describing all possible interactions, sequence diagrams also provide notations for conditionals and
> iterators.

Question sources

Computer Science model exam · 2015 · Question 49

Question 58 | Q00592

Which of the following diagram types is used to represent the static structure of a system?

A. Use case documentation

B. Sequence diagram

C. Class diagram

D. Activity diagram

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Class diagram

Class diagrams describe system structure in classes, attributes, operations and relationships; sequence and activity models describe behavior.

Why each choice is right or wrong

A - Incorrect: Documents functions from actors' viewpoints. See PDF p. 51, 101.

B - Incorrect: Represents interactions over time. See PDF p. 51, 101.

C - Correct: Explicit structural description of classes and objects. See PDF p. 51, 101.

D - Incorrect: Models activities and control flow. See PDF p. 51, 101.

Primary note proof

Study-note section SE17

> A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links. UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process. Structural things describe conceptual or physical structural elements.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 51 | lines 14-24

> 2.3.2. Class Diagrams
> Classes and objects
> Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behavior of a set of objects. Objects are entities that encapsulate state and behavior. Each object has an identity: It can be referred individually and is distinguishable from other objects.
> In UML, classes and objects are depicted by boxes including three compartments. The top compartment displays the name of the class or object. The center compartment displays its attributes, and the bottom compartment displays its operations. The attribute and operation compartment can be omitted for clarity. Object names are underlined to indicate that they are instances. By convention, class names start with an uppercase letter. Objects in object diagrams

Software Engineering module | PDF page 101 | lines 5-6

> In UML, the functional model is represented with use case diagrams, the object model with class diagrams, and the dynamic model with statechart and sequence diagrams.

Question sources

Computer Science model exam · 2015 · Question 61

Question 59 | Q00597

Which is the following is the most important feature of the spiral model?

A. Efficiency management

B. Time management

C. Quality management

D. Risk management

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer D: Risk management.

SE06 makes risk the organizing principle of spiral development. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops an appropriate next result and plans further commitment. Time, efficiency and quality still matter, but they do not distinguish spiral from other processes as directly as its explicit risk-driven cycles.

Why each choice is right or wrong

A - Incorrect: Efficiency is useful, but it is not the defining organizing feature of spiral cycles.

B - Incorrect: Time management matters in many processes and does not distinguish spiral.

C - Incorrect: Quality management is important generally, but the defining spiral emphasis is risk.

D - Correct: Each spiral cycle explicitly evaluates and reduces the most important risks.

Primary note proof

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 96 | lines 12-21

> 4.7. Risk Management
> • Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
> • A risk is a probability that some adverse circumstance will occur
> - Project risks affect schedule or resources;
> - Product risks affect the quality or performance of the software being developed;
> - Business risks affect the organisation developing or procuring the software.

Risk Management Process
- Risk identification- Identify project, product and business risks;
- Risk analysis -Assess the likelihood and consequences of these risks;

Software Engineering module | PDF page 121 | lines 6-15

6.1.4. Software architecture
As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
In this section, we describe a few sample architectures that can be used for different systems. This
is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
representative examples and refer the reader to the literature for more details.

Question sources

Computer Science model exam · 2015 · Question 66

Question 60 | Q00606

_____ is the process of writing down the user and system requirements in a requirements document.

A. Feasibility study

B. Requirements elicitation and analysis.

C. Requirements specification

D. Requirements validation

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Requirements specification

Specification translates analyzed information into a document containing user requirements and detailed system requirements.

Why each choice is right or wrong

A - Incorrect: Decides viability before detailed analysis. See PDF p. 26, 25, 26.

B - Incorrect: Derives requirements rather than being the document-writing step. See PDF p. 26, 25, 26.

C - Correct: Directly names translation into the requirements document. See PDF p. 26, 25, 26.

D - Incorrect: Checks realism, consistency and completeness of the document. See PDF p. 26, 25, 26.

Primary note proof

Study-note section SE08

> Business requirements explain why the project exists and its goals. User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements. A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 26 | lines 10-17

> 3. Requirement’s specification: Requirements specification is the activity of translating the information gathered during the analysis activity in to a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are
> abstract statements of the system requirements for the customer and end-user of the system; system
> requirements are a more detailed description of the functionality to be provided.
> 4. Requirement’s validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered.
> It must then be modified to correct these problems.

Software Engineering module | PDF page 25 | lines 15-18

> There are four main activities in the requirements engineering process:
> 1. Feasibility study: Feasibility shows viability and practicality of a software, project or idea. An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-

Software Engineering module | PDF page 26 | lines 6-9

> 2. Requirement’s elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

Question sources

Computer Science model exam · 2015 · Question 75

Question 61 | Q00612

Incremental model of software development is

A. A reasonable mode when requirements are well define

B. Good model when a working core product is required quickly

C. Good model for the projects with large development teams

D. Good model when user requirements are not complete

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Good model when a working core product is required quickly

Early increments are usable stripped-down products; early working capability is the clearest fit among these options.

Why each choice is right or wrong

A - Incorrect: Well-defined requirements can be useful, but are not the distinctive advantage asked. See PDF p. 22, 22.

B - Correct: Directly supported by early working software. See PDF p. 22, 22.

C - Incorrect: The module specifically notes fewer staff can implement early increments. See PDF p. 22, 22.

D - Incorrect: Feedback can refine requirements, but the module also calls for sufficient overall definition to partition them. See PDF p. 22, 22.

Primary note proof

Study-note section SE04

> Choose incremental delivery when prioritized partial capability must reach users early. Smaller changes can be easier to debug and permit feedback, though architecture and integration still require planning. Iteration need not rebuild the whole system every time, and increments need not be perfectly independent modules. Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 22 | lines 8-12

> This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
> also provide a platform for evaluation by the user.

Software Engineering module | PDF page 22 | lines 24-29

Advantages of Incremental Model:
• Generates working software quickly and early during the software life cycle. This model is more flexible - less costly to change scope and requirements. It is easier to test and debug during a smaller iteration.
• In this model customer can respond to each built. Lowers initial delivery cost.
• Easier to manage risk because risky pieces are identified and handled during its iteration.

Question sources

Computer Science model exam · 2015 · Question 81

Question 62 | Q00620

Which relationship are used when you have one use case that is similar to another, but does something slightly differently or something more.

A. Generalization

B. Aggregation

C. Composition

D. Association

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer A: Generalization

Use-case generalization specializes a general use case by adding detail or varying its realization, as in password versus smart-card authentication.

Why each choice is right or wrong

A - Correct: Direct specialization mechanism described by the module. See PDF p. 51.

B - Incorrect: Whole-part relationship, not use-case specialization. See PDF p. 51.

C - Incorrect: Whole-part lifecycle ownership, not use-case specialization. See PDF p. 51.

D - Incorrect: Communication/linkage alone does not express the stated specialized behavior. See PDF p. 51.

Primary note proof

Study-note section SE16

Generalization relates a specialized actor/use case to a more general one, with a hollow triangle toward the general element. An abstract use case specifies behavior for specialization and is not directly instantiated as a complete use case. It is not defined merely by reuse or lack of a direct actor line. Included behavior is incorporated where the inclusion is reached; do not infer every conditional branch in the base always executes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 51 | lines 4-13

> Generalization relationships
> Generalization/specialization relationships are a third mechanism for reducing the complexity of a model. A use case can specialize a more general one by adding more detail. For example, FieldOfficers are required to authenticate before they can use FRIEND. During early stages of requirements elicitation, authentication is modeled as a high-level Authenticate use case. Later, developers describe the Authenticate use case in more detail and allow for several different hardware platforms. This refinement activity results in two more use cases, AuthenticateWithPasswordwhich enables FieldOfficers to login without any specific hardware, and AuthenticateWithCard, which enables FieldOfficers to login using a smart card. The two new use cases are represented as specializations of the Authenticate use case.

Question sources

Computer Science model exam · 2015 · Question 89

Question 63 | Q00649

_____a software product is necessary to rectify the bugs observed while the system is in use.

A. Adaptive maintenance

B. Corrective maintenance

C. Perfective maintenance

D. Preventive maintenance

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer B: Corrective maintenance

The purpose of the change determines the maintenance category. Corrective maintenance repairs faults observed during use. Adaptive maintenance handles environmental changes, perfective maintenance improves useful capability or performance, and preventive maintenance reduces future risk before a corresponding failure occurs. Because the scenario starts with observed bugs that need repair, corrective maintenance is the direct category rather than one of the other three motives.

Why each choice is right or wrong

A - Incorrect: Adaptive maintenance responds to a changed environment.

B - Correct: Corrective maintenance repairs observed defects.

C - Incorrect: Perfective maintenance improves useful capability or performance.

D - Incorrect: Preventive maintenance reduces future risk rather than primarily fixing the stated observed bugs.

Primary note proof

Study-note section SE40

> Corrective maintenance fixes faults; adaptive maintenance responds to environment changes; perfective maintenance improves useful capability/performance; preventive maintenance reduces future risk, such as restructuring fragile code. Motive determines category; refactoring can support preventive or perfective work. Maintenance often lasts years and includes understanding legacy code and regression effort, so it may dominate lifecycle cost.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 19 | lines 24-29

> 5. Operation and maintenance: Normally (although not necessarily), this is the longest life cycle phase. Maintenance of a typical software product requires much more than the effort necessary to
> develop the product itself. The system is installed and put into practical use. Maintenance involves
> correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system ‘s services as new requirements are discovered.

Question sources

Computer Science model exam · 2015 · Question 118

Question 64 | Q00677

Criteria that should not be considered when evaluating or assessing requirement analysis?

A. Correctness

B. Need assessment

C. Consistency

D. Speed

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: Speed

Validation checks correctness, consistency and realism against stakeholder needs. Speed is a property of the future system's performance, not a general criterion for whether the analysis is valid.

Why each choice is right or wrong

A - Incorrect: Explicit requirements-validation property. See PDF p. 68, 79.

B - Incorrect: Assessing actual needs supports faithful requirements. See PDF p. 68, 79.

C - Incorrect: Explicit absence-of-contradiction property. See PDF p. 68, 79.

D - Correct: System response speed is a nonfunctional requirement, not an analysis quality criterion itself. See PDF p. 68, 79.

Primary note proof

Study-note section SE11

> Correct requirements reflect actual stakeholder needs. Complete requirements address the agreed scope, normal and exceptional behavior. Consistency excludes contradictions. Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable within constraints. Verifiability means objective evidence can check fulfillment. Replace “fast for most users” with a defined metric, workload and threshold.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 68 | lines 18-26

> system specification. Requirement validation involves checking if the specification is correct, complete, consistent, unambiguous, and realistic. A specification is correct if it represents the client’s view of the system (i.e., everything in the requirements model accurately represents an aspect of the system). It is complete if all possible scenarios through the system are described, including exceptional behavior (i.e., all aspects of the system are represented in the requirements
> model). The system specification is consistent if it does not contradict itself. The system specification is unambiguous if exactly one system is defined (i.e., it is not possible to interpret the specification two or more different ways). Finally, it is realistic if the system can be implemented within constraints. These properties are illustrated with UML instance diagrams in

Software Engineering module | PDF page 79 | lines 3-9

> • Performance characteristics. How responsive should the system be? How many concurrent users should it support? What is a typical or extreme load?
> • Error handling and extreme conditions. How should the system handle exceptions? Which exceptions should the system handle? What is the worse environment in which the system is expected to perform? Are there safety requirements on the system?
> • Quality issues. How reliable/available/robust should the system be? What is the client’s involvement in assessing the quality of the system or the development process?

Question sources

Computer Science model exam 2 · Question 18

Question 65 | Q00739

A measurement that defines the level of inter-dependability among modules of a programis__.

A. Cohesive

B. Coupling

C. Design

D. Testing

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Coupling

Coupling measures dependency between subsystems; coherence/cohesion concerns dependency within one subsystem.

Why each choice is right or wrong

A - Incorrect: Internal relatedness, not inter-module dependency. See PDF p. 120, 120.

B - Correct: Exact strength-of-dependencies measure between subsystems. See PDF p. 120, 120.

C - Incorrect: An activity that chooses structures, not this measurement. See PDF p. 120, 120.

D - Incorrect: An error-detection activity, not this measurement. See PDF p. 120, 120.

Primary note proof

Study-note section SE22

> Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module's internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 120 | lines 6-12

> 6.1.3. Coupling and coherence
> Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
> is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Software Engineering module | PDF page 120 | lines 24-27

Coherence is the strength of dependencies within a subsystem. If a subsystem contains many objects that are related to each other and perform similar tasks, its coherence is high. If a subsystem
contains a number of unrelated objects, its coherence is low. A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence.

Question sources

Computer Science model exam 2 · Question 80

Question 66 | Q00740

Which are supported modularization benefits? Select all that apply. Read D as a claim that modularization alone guarantees security.

A. Smaller components are difficult to maintain

B. Components with high cohesion can be re-used again

C. Program can be divided based on functional aspects

D. Modularization alone guarantees security.

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B, C: Components with high cohesion can be re-used again; Program can be divided based on functional aspects

Cohesive components with clear interfaces can be reused, and functional decomposition divides responsibilities into manageable modules. Smaller modules can also improve understanding and maintenance, so A reverses the ordinary benefit. Security requires appropriate protection and boundary enforcement; simply dividing code into modules does not guarantee it. Thus B and C are supported structural benefits without the unconditional security promise.

Why each choice is right or wrong

A - Incorrect: This reverses the ordinary maintenance benefit of manageable cohesive modules.

B - Correct: Cohesive components can be reused in suitable contexts.

C - Correct: Functional decomposition separates related responsibilities into modules.

D - Incorrect: Module boundaries alone do not guarantee security.

Primary note proof

Study-note section SE22

Logical cohesion groups similar-category operations, often selected by a flag; temporal cohesion groups operations done at the same time; functional cohesion focuses on one task. Splitting an unrelated three-purpose component improves modularity. If behavior is preserved, the restructuring is refactoring; the words are not mutually exclusive answers. Smaller modules can

aid understanding, independent work, reuse and maintenance, but too many interfaces also create complexity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 118 | lines 11-16

6.1.1. Subsystems and classes
In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Software Engineering module | PDF page 120 | lines 24-27

Coherence is the strength of dependencies within a subsystem. If a subsystem contains many objects that are related to each other and perform similar tasks, its coherence is high. If a subsystem
contains a number of unrelated objects, its coherence is low. A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence.

Software Engineering module | PDF page 32 | lines 12-21

A class in an object-oriented system carefully delineates between its interfaces the routines and attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact
on other classes, and so, the impact is minimized. However, the object-oriented approach is not a panacea; nothing is magical here that will promote perfect design or perfect code.
Promotion of reusability
Objects are reusable because they are modeled directly out of a real-world problem domain. Each
object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.

Question sources

Computer Science model exam 2 · Question 81

Question 67 | Q00741

Which of the following is not element of Requirement Engineering Process?

A. Software Requirement Specification

B. Feasibility Steady

C. Software Requirement Design

D. Requirement Gathering

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Software Requirement Design

The four listed requirements-engineering activities are feasibility, elicitation/analysis, specification and validation. Software requirement design is not one of them.

Why each choice is right or wrong

A - Incorrect: Specification is explicitly included. See PDF p. 25, 26, 26.

B - Incorrect: Feasibility study is explicitly included, despite the typo Steady. See PDF p. 25, 26, 26.

C - Correct: Designing the solution is separate from the listed requirements-engineering activities. See PDF p. 25, 26, 26.

D - Incorrect: Gathering is part of elicitation/analysis. See PDF p. 25, 26, 26.

Primary note proof

Study-note section SE08

> Business requirements explain why the project exists and its goals. User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements. A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 25 | lines 15-18

> There are four main activities in the requirements engineering process:
> 1. Feasibility study: Feasibility shows viability and practicality of a software, project or idea. An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-

Software Engineering module | PDF page 26 | lines 10-17

> 3. Requirement’s specification: Requirements specification is the activity of translating the information gathered during the analysis activity in to a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are
> abstract statements of the system requirements for the customer and end-user of the system; system
> requirements are a more detailed description of the functionality to be provided.
> 4. Requirement’s validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered.
> It must then be modified to correct these problems.

Software Engineering module | PDF page 26 | lines 6-9

> 2. Requirement’s elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

Question sources

Computer Science model exam 2 · Question 82

Question 68 | Q00742

Which type of test determine if the system meets the entire requirements (functional and nonfunctional)

A. Unit Test

B. Integration test

C. System test

D. Acceptance test

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: System test

System testing examines the complete system's functional and nonfunctional requirements. Acceptance is a client decision against the agreement, which may also cover requirements but is a more specific acceptance purpose.

Why each choice is right or wrong

A - Incorrect: Isolates objects/components. See PDF p. 144, 147, 145.

B - Incorrect: Finds faults among combined components. See PDF p. 144, 147, 145.

C - Correct: Explicit complete-system functional/nonfunctional scope. See PDF p. 144, 147, 145.

D - Incorrect: Client agreement testing, rather than the general full-system test scope asked. See PDF p. 144, 147, 145.

Primary note proof

Study-note section SE35

> Acceptance testing supports the customer/user’s acceptance decision against agreed needs and criteria. Alpha typically happens in a controlled developer setting with selected users; beta uses external users in their environments. None of these levels is universally the longest. Unit is a level, while white-box is a technique; a unit can be tested using black-box cases too.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Software Engineering module | PDF page 147 | lines 14-18

• Unit testing, which finds faults by isolating an individual component using test stubs and drivers and by exercising the component using a test case
• Integration testing, which finds faults by integrating several components together
• System testing, which focuses on the complete system, its functional and nonfunctional requirements, and its target environment

Software Engineering module | PDF page 145 | lines 3-5

o Acceptance testing and installation testing check the requirements against the project agreement and should be done by the client, if necessary with support by the developers.

Question sources

Computer Science model exam 2 · Question 83

Question 69 | Q00744

All of the following are required to develop an effective test except:

A. Detailed understanding of the system

B. Application and solution domain knowledge

C. Knowledge of the testing techniques

D. Skill to specific programming language

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer D: Skill to specific programming language

Black-box equivalence tests can be derived from inputs and expected behavior without writing in a particular implementation language. Domain/system knowledge and testing methods remain relevant.

Why each choice is right or wrong

A - Incorrect: Needed to choose meaningful cases and expected results. See PDF p. 149, 149.

B - Incorrect: Needed to interpret required behavior. See PDF p. 149, 149.

C - Incorrect: Needed to partition inputs or choose boundaries/paths appropriately. See PDF p. 149, 149.

D - Correct: Not universally necessary for behavior-based test design; source-language knowledge applies particularly to white-box work. See PDF p. 149, 149.

Primary note proof

Study-note section SE36

> Functional tests check required behavior against the SRS and acceptance criteria. Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits. Usability, security, compatibility and reliability tests target their respective qualities. “Normal operating conditions” alone does not distinguish functional from performance testing.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 6-10

> Equivalence testing is a black box testing technique that minimizes the number of test cases. The possible inputs are partitioned into equivalence classes, and a test case is selected for each class. The assumption of equivalence testing is that systems usually behave in similar ways for all members of a class. To test the behavior associated with an equivalence class, we only need to test
> one member of the class. Equivalence testing consists of two steps: identification of the

Software Engineering module | PDF page 149 | lines 24-29

> Path testing
> Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path testing is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.
> The starting point for path testing is the flow graph. A flow graph consists of nodes representing

Question sources

Computer Science model exam 2 · Question 85

Question 70 | Q00745

Use case description commonly contain:

A. Trigger, associations, guarantee.

B. Precondition, generalizations, main scenario.

C. Precondition, main scenario, alternative scenarios

D. None of the above

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Precondition, main scenario, alternative scenarios

A use-case description includes entry conditions, the numbered normal flow and exceptional cases.

Why each choice is right or wrong

A - Incorrect: Trigger/associations/guarantee are not its stated three-part summary. See PDF p. 48.

B - Incorrect: Generalization is a relationship, not one of these template fields. See PDF p. 48.

C - Correct: Matches preconditions and normal/exceptional flow content. See PDF p. 48.

D - Incorrect: A supported summary is offered. See PDF p. 48.

Primary note proof

Study-note section SE15

> Example: Submit Incident. Precondition: officer authenticated. 1 Officer supplies location and description. 2 System validates details. 3 System records the incident. 4 System returns its ID. Alternative 2a: location missing → report the problem and request correction; save nothing. Success postcondition: incident and ID persist; failure guarantee: no partial record. An alternate flow is a conditional variation; an exception flow handles a failure. Preconditions hold before entry, while a trigger starts the use case.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 48 | lines 18-32

> To describe a use case, we use a template composed of six fields (see also Figure 2.2):
> • The name of the use case is unique across the system so that developers (and project participants) can unambiguously refer to the use case.
> • Participating actors are actors interacting with the use case.

• Entry conditions describe the conditions that need to be satisfied before the use case is initiated.
• The flow of events describes the sequence of actions of the use case, which are be numbered for reference. The common case (i.e., cases that occur frequently) and the exceptional cases (i.e., cases that seldom occur, such as errors and unusual conditions) are described separately in different use cases for clarity.
• Exit conditions describe the conditions that are satisfied after the completion of the use case.
• Special requirements are requirements that are not related to the functionality of the system. These include constraints on the performance of the system, its implementation, the hardware platforms it runs on, and so on.

Question sources

Computer Science model exam 2 · Question 86

Question 71 | Q00746

Aggregation, composition, generalization. Which document is created by a system analyst after the requirements are collected from different stakeholders?

A. Requirement gathering

B. Software Requirement specification

C. Software Requirement testing

D. Software Requirement validation

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Software Requirement specification

After collection, requirements are organized into the SRS. The opening aggregation/composition/generalization fragment does not alter the requested document.

Why each choice is right or wrong

A - Incorrect: A gathering activity, not the resulting specification document. See PDF p. 26, 83.

B - Correct: The defined organized requirements document. See PDF p. 26, 83.

C - Incorrect: Testing is an activity rather than this document. See PDF p. 26, 83.

D - Incorrect: Validation checks a specification rather than naming its document. See PDF p. 26, 83.

Primary note proof

Study-note section SE08

Business requirements explain why the project exists and its goals. User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements. A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 26 | lines 10-17

3. Requirement's specification: Requirements specification is the activity of translating the information gathered during the analysis activity in to a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are
abstract statements of the system requirements for the customer and end-user of the system; system
requirements are a more detailed description of the functionality to be provided.
4. Requirement's validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered.
It must then be modified to correct these problems.

Software Engineering module | PDF page 83 | lines 12-25

3.4.4. Documenting Requirements Elicitation
The results of the requirements elicitation activity and the analysis activity are documented in the
Requirements Analysis Document (RAD). This document completely describes the system in terms of functional and nonfunctional requirements and serves as a contractual basis between the
client and the developers. The audience for the RAD includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design). The first part of the document, including use cases and nonfunctional requirements, is written during requirements
elicitation. The formalization of the specification in terms of object models is written during analysis.
The first section of the RAD is an Introduction. Its purpose is to provide a brief overview of the function of the system and the reasons for its development, its scope, and references to the development context (e.g., related problem statement, references to existing systems, feasibility studies). The introduction also includes the objectives and success criteria of the project.

Question sources

Computer Science model exam 2 · Question 87

Question 72 | Q00749

_____is a process which translate system specification into a formal or semi formal model.

A. Implementation

B. Elicitation

C. Analysis

D. Design

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Analysis

Analysis structures and formalizes the natural-language system specification into a formal or semiformal analysis model.

Why each choice is right or wrong

A - Incorrect: Converts design to executable code. See PDF p. 100, 62.

B - Incorrect: Produces the initial specification with stakeholders. See PDF p. 100, 62.

C - Correct: Exact formalization activity described. See PDF p. 100, 62.

D - Incorrect: Derives internal solution structure after analysis. See PDF p. 100, 62.

Primary note proof

Study-note section SE13

A scenario is one concrete execution; a use case abstracts the related successful and alternative scenarios. Analysis integrates functional (use cases/scenarios), structural/object (classes) and dynamic (states/interactions) models. Entity objects represent domain information such as Incident; boundary objects mediate interaction, such as ReportForm; control objects coordinate a use case. A UI form is normally boundary, not entity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 100 | lines 11-24

Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users
Figure 4.1 Products of requirements elicitation and analysis (UML activity diagram).
Althoughtheanalysismodelmaynotbeunderstandabletotheusersandtheclient, it helps developers verify the system specification produced during requirements elicitation. There is a natural tendency for users and developers to postpone difficult decisions until later in the project. A decision may be difficult because of lack of domain knowledge, lack of technological knowledge,

or simply because of disagreements among users and developers. Postponing decisions enables the
project to move on smoothly and avoid confrontation with reality or peers. Unfortunately, difficult
decisions will eventually need to be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation. Translating a system specification in to a formal or semi-formal model forces developers to identify and resolve difficult issues early in
the development.

Software Engineering module | PDF page 62 | lines 9-16

Requirement's elicitation focuses on describing the purpose of the system. The client, the developers, and the users identify a problem area and define a system that addresses the problem.
Such a definition is called a system specification and serves as a contract between the client and the developers. The system specification is structured and formalized during analysis to produce an analysis model (see Figure 3.1). Both system specification and analysis model represent the same information. They differ only in the language and notation they use. The system specification
is written in natural language, whereas the analysis model is usually expressed in a formal or semiformal notation.

Question sources

Computer Science model exam 2 · Question 90

Question 73 | Q00750

Which maintenance risk can arise from poorly designed objects with complex or poorly separated responsibilities?

A. Easier maintenance

B. Objects may be understood as stand-alone entities

C. Objects are potentially reusable components

D. Poorly designed objects can be difficult to maintain.

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: Poorly designed objects can be difficult to maintain.

Encapsulation and reuse can ease maintenance when objects have clear responsibilities and interfaces. Poor object design can still be complex, so its objects may become difficult to maintain. That conditional risk is D. The other choices describe potential advantages of well-designed objects rather than disadvantages arising from the stated poor design. Object orientation alone does not guarantee either maintenance outcome.

Why each choice is right or wrong

A - Incorrect: Easier maintenance is a potential benefit, not this risk.

B - Incorrect: Understanding encapsulated objects through their interfaces is a potential benefit.

C - Incorrect: Potential reuse is an advantage of suitable design.

D - Correct: Poorly designed complex objects can be difficult to maintain.

Primary note proof

Study-note section SE22

> Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module's internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Study-note section SE01

> Procedural decomposition organizes functions; object oriented decomposition organizes objects with state and behavior. Either can use iterative or sequential development. OOP does not automatically shorten every project. Reuse, encapsulation and consistent models can reduce duplication and ease change; poor object design can still be complex. Functional programming emphasizes function evaluation and controlled side effects; event-driven programs react to events.

Study-note section SE22

> Splitting an unrelated three-purpose component improves modularity. If behavior is preserved, the restructuring is refactoring; the words are not mutually exclusive answers.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 31 | lines 19-31

> 1.3.1. Why an object oriented
> • Object oriented systems are
> o Easier to adapt to changes
> o More robust
> o Easier to maintain
> o Promote greater design and code reuse
> o Creates modules of functionality
> • REASONS FOR WORKING OF OBJECT ORIENTED SYSTEMS:

Higher level of abstraction
The object-oriented approach supports abstraction at the object level. Since objects encapsulate both data (attributes) and functions (methods), they work at a higher level of abstraction. The development can proceed at the object level and ignore the rest of the system for as long as necessary. This makes designing, coding, testing, and maintaining the system much simpler.

Question sources

Computer Science model exam 2 · Question 91

Question 74 | Q00897

In a class diagram, a filled diamond is at Car's end of its Engine association. In this model an Engine remains owned by its Car for its entire lifetime, with no detachment or transfer. What does the relationship imply?

A. Car uses Engine temporarily

B. Engine can exist independently of the Car

C. Engine is part of Car and cannot exist independently under the stated lifetime policy.

D. Car inherits from Engine

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Engine is part of Car and cannot exist independently under the stated lifetime policy.

The filled diamond at Car denotes composition: Car is the whole and Engine is its strongly owned part. Composition destroys parts that remain owned when the whole is destroyed. Because this model forbids detachment or transfer, the Engine's lifetime cannot continue independently of its Car. Temporary use would be a dependency, and inheritance would use a hollow generalization triangle instead.

Why each choice is right or wrong

A - Incorrect: Temporary use is a dependency, not composition.

B - Incorrect: The stipulated permanent ownership excludes an independent Engine lifetime.

C - Correct: Engine is the owned part and cannot outlive Car under the stated policy.

D - Incorrect: Inheritance uses a hollow triangle rather than a filled diamond.

Primary note proof

Study-note section SE18

Composition uses a filled diamond at the whole end and strong ownership: a part is in at most one composite at a time, and parts still owned are destroyed with that composite. Parts may be

detached/transferred under a defined model. Thus “cannot ever exist independently” is an oversimplification. For Car composing Engine, the diamond belongs beside Car; saying it points to Engine is imprecise. The Car-Engine example describes composition.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 51 | lines 14-24

2.3.2. Class Diagrams
Classes and objects
Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behavior of a set of objects. Objects are entities that encapsulate state and behavior. Each object has an identity: It can be referred individually and is distinguishable from other objects.
In UML, classes and objects are depicted by boxes including three compartments. The top compartment displays the name of the class or object. The center compartment displays its attributes, and the bottom compartment displays its operations. The attribute and operation compartment can be omitted for clarity. Object names are underlined to indicate that they are instances. By convention, class names start with an uppercase letter. Objects in object diagrams

Software Engineering module | PDF page 53 | lines 19-28

Aggregation
Associations are used to represent a wide range of connections among a set of objects. In practice,
a special case of association occurs frequently: composition. For example, a State contains many Counties, which in turn contains many Townships. A PoliceStation is composed of PoliceOfficers. Another example is a Directory that contains a number of Files. Such relationships could be modeled using a one-to-many association. Instead, UML provides the concept of an aggregation to denote composition. An aggregation is denoted by a simple line with a diamond at the container
end of the association. Although one-to-many associations and aggregations can be used alternatively, aggregations are preferable because they emphasize the hierarchical aspects of the relationship.

Question sources

Computer Science Exit Exam · 2018 · Question 10

Question 75 | Q00942

From the following statement which one is disadvantage of waterfall model?

A. model is simple and easy to understand and use

B. It is easy manage due to the rigidity of the model

C. .
Not suitable for projects where requirements are at a moderate to high risk of changing

D. In this model phases are processed and completed one at a time

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: .
Not suitable for projects where requirements are at a moderate to high risk of changing

Waterfall requires early stable requirements and has difficulty responding to change, so moderate-to-high change risk is its documented disadvantage.

Why each choice is right or wrong

A - Incorrect: Listed as an advantage. See PDF p. 20.

B - Incorrect: Listed as an advantage from distinct deliverables/reviews. See PDF p. 20.

C - Correct: Directly matches the disadvantage on PDF p.21. See PDF p. 20.

D - Incorrect: Sequential non-overlap is a structural feature, not the stated change-risk disadvantage. See PDF p. 20.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 20 | lines 18-22

> the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
> In principle, the waterfall model should only be used when the requirements are well

understood
and unlikely to change radically during system development. However, the waterfall model

Question sources

Computer Science Exit Exam · 2018 · Question 55

Question 76 | Q00948

Which statement specifically constrains response-time performance under a stated workload?

A. The system allow user to generate PDF report.

B. The system respond to user's request within 2 second under normal load

C. .
The system allow user to store data on relational database

D. The system allow user to submit emergency report

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: The system respond to user's request within 2 second under normal load

The two-second response obligation bounds the speed of a service under the stated load, so it is a performance requirement. Generating a PDF and submitting an emergency report describe services the system provides. The database clause describes storage behavior together with an implementation choice; it does not impose a response-time threshold. The explicit performance scope removes the broader taxonomy overlap between nonfunctional requirements and technology constraints.

Why each choice is right or wrong

A - Incorrect: Report generation specifies a service, not its response-time limit.

B - Correct: A two-second response bound under normal load is a quantitative speed target.

C - Incorrect: This specifies storage behavior and a database technology, without a timing bound.

D - Incorrect: Submitting an emergency report specifies a function rather than its speed.

Primary note proof

Study-note section SE10

Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning. Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Study-note section SE10

"Search responds within two seconds under 100 concurrent users" is measurable performance. "New users finish checkout within one minute without help" is usability. "Use Java" is a technology/pseudo-requirement, often separately classified as a system constraint. Dependability includes reliability, safety and robustness; portability is not its usual central category. Performance degradation at peak load primarily concerns performance, not automatically reliability.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 65 | lines 19-23

3.2.1. Functional Requirements
Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. For example, the following is an example of functional requirements for SatWatch, a watch that resets itself without user intervention:

Software Engineering module | PDF page 66 | lines 14-18

3.2.2. Non-Functional and Pseudo-Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Nonfunctional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands)
or accuracy (i.e., how precise are the system's numerical answers). The following are the

Software Engineering module | PDF page 67 | lines 3-8

Pseudo-requirements are requirements imposed by the client that restrict the implementation of the system. Typical pseudo-requirements are the implementation language and the platform on which the system is to be implemented. For life-critical developments, pseudo-requirements often
include process and documentation requirements (e.g., the use of a formal specification method, the complete release of all work products). Pseudo-requirements have usually no direct effect on the users' view of the system. The following are the pseudo functional requirements for SatWatch:

Question sources

Computer Science Exit Exam · 2018 · Question 61

Question 77 | Q00949

In object design, what does the contract typically includes?

A. The class diagram and sequence diagram

B. Project budget and timeline

C. Invariants, preconditions, post conditions

D. The use case names and actors role

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Invariants, preconditions, post conditions

Contracts establish class-wide invariants and caller/callee obligations before and after an operation.

Why each choice is right or wrong

A - Incorrect: Diagrams can document a design but are not the contract constraint categories. See PDF p. 136.

B - Incorrect: Project planning terms, not class contracts. See PDF p. 136.

C - Correct: Exactly the three contract constraint types. See PDF p. 136.

D - Incorrect: Use-case identifiers and roles do not specify class invariants and operation obligations. See PDF p. 136.

Primary note proof

Study-note section SE25

> A contract states preconditions, postconditions and invariants. For withdraw(amount): require amount>0 and amount<=balance; ensure balance=oldBalance-amount; preserve balance>=0. Validation checks inputs; implementation enforces the transition. A complete feature needs both UI/API and persisted behavior, not only a diagram.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 136 | lines 3-11

> • An invariant is a predicate that is always true for all instances of a class. Invariants are constraints associated with classes or interfaces. Invariants are used to specify consistency constraints among class attributes.
> • A precondition is a predicate that must be true before an operation is invoked. Preconditions are associated with a specific operation. Preconditions are used to specify constraints that a caller must meet before calling an operation.
> • A post-condition is a predicate that must be true after an operation is invoked. Post-conditions are associated with a specific operation. Post-conditions are used to specify constraints that the object must ensure after the invocation of the operation.

Question sources

Computer Science Exit Exam · 2018 · Question 62

Question 78 | Q00967

Which one of the following UML building block that defines the static part of the model and represents physical and conceptual elements?

A. Structural things

B. Behavioral things

C. ¢.
An notational things

D. Grouping things

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer A: Structural things

Structural things explicitly define the static model and physical/conceptual elements.

Why each choice is right or wrong

A - Correct: Matches the module verbatim. See PDF p. 45, 45, 46.

B - Incorrect: Behavioral things are dynamic. See PDF p. 45, 45, 46.

C - Incorrect: Annotational things capture notes and remarks. See PDF p. 45, 45, 46.

D - Incorrect: Grouping things organize elements into packages. See PDF p. 45, 45, 46.

Primary note proof

Study-note section SE17

> A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links. UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process. Structural things describe conceptual or physical structural elements.

Study-note section SE-S03

> In the course's UML building-block terminology, structural things describe static conceptual or physical elements. Behavioral things describe dynamic interactions and state behavior. Grouping things organize model elements into packages. Annotational things attach explanatory remarks or constraints, usually as notes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 45 | lines 3-17

> Structural things:
> The Structural things define the static part of the model. They represent physical and conceptual

elements. Following are the brief descriptions of the structural things.
Class:
Class represents set of objects having similar responsibilities.
Interface:
Interface defines a set of operations which specify the responsibility of a class.
Collaboration:
Collaboration defines interaction between elements.
Use case:
Use case represents a set of actions performed by a system for a specific goal.
Component:
Component describes physical part of a system.
Node:
A node can be defined as a physical element that exists at run time.

Software Engineering module | PDF page 45 | lines 18-23

Behavioral things:
A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things:
Interaction:
Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.

Software Engineering module | PDF page 46 | lines 7-16

Grouping things:
Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available:
Package:
Package is the only one grouping thing available for gathering structural and behavioral things.
An notational things:
An notational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. Note is the only one An notational thing available.
Note:
A note is used to render comments, constraints etc. of an UML element.

Question sources

Computer Science Exit Exam · 2018 · Question 80

Question 79 | Q00971

After deploying a new software system, users reports that the system slows during peak hour. Which software quality attribute primarily affected?

A. Maintainability

B. Performance

C. Reliability

D. Usability

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Performance

Slow response at peak load concerns responsiveness and concurrency/load constraints, hence performance.

Why each choice is right or wrong

A - Incorrect: Concerns cost/ease of modifying the system. See PDF p. 79, 66.

B - Correct: Directly affected response-speed/load quality. See PDF p. 79, 66.

C - Incorrect: Reliability concerns conformance/failures; slow service could violate requirements, but performance is the specific attribute. See PDF p. 79, 66.

D - Incorrect: Usability concerns ease of learning/using rather than primarily load-dependent response speed. See PDF p. 79, 66.

Primary note proof

Study-note section SE10

> “Search responds within two seconds under 100 concurrent users” is measurable performance. “New users finish checkout within one minute without help” is usability. “Use Java” is a technology/pseudo-requirement, often separately classified as a system constraint. Dependability includes reliability, safety and robustness; portability is not its usual central category. Performance degradation at peak load primarily concerns performance, not automatically reliability.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 79 | lines 3-9

> • Performance characteristics. How responsive should the system be? How many concurrent users should it support? What is a typical or extreme load?
> • Error handling and extreme conditions. How should the system handle exceptions? Which exceptions should the system handle? What is the worse environment in which the system is expected to perform? Are there safety requirements on the system?

• Quality issues. How reliable/available/robust should the system be? What is the client's involvement in assessing the quality of the system or the development process?

Software Engineering module | PDF page 66 | lines 14-18

3.2.2. Non-Functional and Pseudo-Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Nonfunctional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands)
or accuracy (i.e., how precise are the system's numerical answers). The following are the

Question sources

Computer Science Exit Exam · 2018 · Question 84

Question 80 | Q01197

Which choice most completely lists candidate heuristics for grouping objects into subsystems? Treat them as starting guidance, not mandatory boundaries.

A. Assign objects identified in one use case into the same subsystem.

B. Create a dedicated subsystem for objects used for moving data among subsystems.

C. Minimize the number of associations crossing subsystem boundaries.

D. All of the above

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: All of the above

SE21 recommends grouping related functionality, minimizing cross-subsystem associations and isolating communication infrastructure where useful. Grouping objects serving one use case can be a starting heuristic, though workflows do not dictate every boundary. B and C directly support isolation and low coupling. Since the question asks what can be a heuristic, the inclusive option is appropriate.

Why each choice is right or wrong

A - Incorrect: True as one component, but incomplete as the requested full selection. Objects collaborating on one use case can be grouped as a starting heuristic, subject to other design constraints.

B - Incorrect: True as one component, but incomplete as the requested full selection. Communication infrastructure can be isolated in a dedicated subsystem when useful.

C - Incorrect: True as one component, but incomplete as the requested full selection. Reducing cross-boundary associations reduces coupling between subsystems.

D - Correct: All three can guide decomposition, provided use-case grouping is not treated as an absolute rule.

Primary note proof

Study-note section SE21

> System design turns the analysis model into architecture: design goals, subsystem responsibilities, dependencies, hardware mapping, data management, access control, control flow and startup/shutdown/error policies. Detail/object design refines APIs, parameter/return types, exceptions, component choices and algorithms. A subsystem is a manageable logical collection with services and an interface; it need not be a physical replaceable artifact.

Study-note section SE21

> Group closely related objects and functionality, minimize associations crossing boundaries, and isolate communication infrastructure where useful.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 118 | lines 11-16

> 6.1.1. Subsystems and classes
> In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Software Engineering module | PDF page 116 | lines 16-29

> The analysis model describes the system completely from the actors' point of view and serves as the basis of communication between the client and the developers. The analysis model, however, does not contain information about the internal structure of the system, its hardware configuration,
> or, more generally, how the system should be realized. System design is the first step in this direction. System design results in the following products:
> • List of design goals, describing the qualities of the system that developers should optimize
> • Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.
> The design goals are derived from the nonfunctional requirements. Design goals guide the decisions to be made by the developers especially when trade-offs are needed. The subsystem decomposition constitutes the bulk of system design. Developers divide the system into manageable pieces to deal with complexity: Each subsystem is assigned to a team and realized independently. In order for this to be possible, though, developers need to address system-wide

Software Engineering module | PDF page 120 | lines 6-12

6.1.3. Coupling and coherence
Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Question sources

Computer Science model exam 2 · Question 18

Question 81 | Q01205

Which statement is false about a logical software subsystem? Interpret complexity reduction as managing solution-design complexity, not changing the inherent application domain.

A. A physical and replaceable part of the system.

B. It helps manage the complexity of the solution design.

C. Decompose a system into simpler parts.

D. Well-defined interfaces that encapsulate the state and behavior of its contained classes.

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer A: A physical and replaceable part of the system.

A subsystem is a manageable logical collection of responsibilities with services and an interface. It need not be a physical replaceable artifact, so A is false as a definition. Decomposing the designed solution into simpler groups helps manage complexity, and defined interfaces can encapsulate implementation state and behavior. These design benefits concern the solution organization rather than reducing the inherent complexity of its problem domain.

Why each choice is right or wrong

A - Correct: A logical subsystem need not be physical or independently replaceable.

B - Incorrect: Managing solution-design complexity is a supported decomposition benefit.

C - Incorrect: Decomposition groups the system into simpler responsibilities.

D - Incorrect: Defined interfaces encapsulate contained implementation details.

Primary note proof

Study-note section SE21

Group closely related objects and functionality, minimize associations crossing boundaries, and isolate communication infrastructure where useful. User workflows inform decomposition but do not alone dictate the internal structure. Analysis describes domain needs; subsystem decomposition is principally a design result. Architecture documentation supports communication and records major decisions, not every source-level execution step.

Study-note section SE21

A subsystem is a manageable logical collection with services and an interface; it need not be a physical replaceable artifact.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 118 | lines 11-16

6.1.1. Subsystems and classes
In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Software Engineering module | PDF page 119 | lines 13-25

6.1.2. Services and subsystem interfaces
A subsystem is characterized by the services it provides to other subsystems. A service is a set of related operations that share a common purpose. A subsystem providing a notification service, for
example, defines operations to send notices, look up notification channels, and subscribe and unsubscribe to a channel.
The set of operations of a subsystem that are available to other subsystems form the subsystem interface. The subsystem interface, also referred to as the application programmer interface (API),
includes the name of the operations, their parameters, their types, and their return values. System
design focuses on defining the services provided by each subsystem that is, enumerating the operations, their parameters, and their high-level behavior. Object design will focus on defining the subsystem interfaces, that is, the type of the parameters and the return value of each operation.
The definition of a subsystem in terms of the services it provides helps us focus on its interface as opposed to its implementation. A good subsystem interface should provide as little information

Question sources

Computer Science model exam 2 · Question 26

Question 82 | Q01208

Which of the following
is an example of Usability requirements?

A. A new user shall complete the main interface task within one minute without help.

B. Robustness

C. Mean time to failure

D. Adaptability

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: A new user shall complete the main interface task within one minute without help.

A usability requirement states an observable target for successful and learnable use. Requiring a new user to finish a main interface task without help within one minute gives a task, user context, success condition and threshold. Robustness concerns dependable behavior under adverse conditions, mean time to failure is a reliability measure, and adaptability concerns change. Merely naming a user interface would not itself specify an obligation, so the option supplies a genuine requirement.

Why each choice is right or wrong

A - Correct: The task-success and unaided-learning target is a measurable usability obligation.

B - Incorrect: Robustness concerns dependable behavior under adverse conditions rather than ease of learning a task.

C - Incorrect: Mean time to failure measures reliability rather than user task ease.

D - Incorrect: Adaptability concerns accommodating change rather than the stated user task target.

Primary note proof

Study-note section SE10

> Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning.

Study-note section SE10

> “New users finish checkout within one minute without help” is usability.

Study-note section SE-S07

> Robustness concerns reliable operation across varied or difficult operating conditions and controlled failure outside the supported range. It is different from usability, which concerns

people successfully learning and carrying out tasks. Improving an interface's task success does not by itself establish robustness under adverse conditions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 78 | lines 12-30

3.3.7. Identifying Non-Functional Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related
to the functional behavior of the system. Nonfunctional requirements span a number of issues, from user interface look and feel to response time requirements to security issues.
Nonfunctional
requirements are defined at the same time as functional requirements are, because they have as much impact on the development and cost of the system.
For example, consider a mosaic display that an air traffic controller uses to track planes. A mosaic
display system complies data from a series of radar and databases (hence the term "mosaic") into
a summary display indicating all aircraft in a certain area, including their identification, speed, and
altitude. The number of aircraft such a system can display constrains the performance of the air traffic controller and the cost of the system. If the system can only handle a few aircraft simultaneously, the system cannot be used at busy airports. On the other hand, a system able to handle a large number of aircraft is more costly and more complex to build.
Nonfunctional requirements can be elicited by investigating the following issues.
• User interface and human factors. What kind of interface should the system provide? What is the level of expertise of the users?
• Documentation. What level of document is required? Should only user documentation be provided? Should there be technical documentation for maintainers? Should the development process be documented?

Software Engineering module | PDF page 82 | lines 3-12

though this type of approach could be used for any parameter, usability tests focus on usability parameters, such as the ease to learn the system, the time to accomplish a task, or the rate of errors
a user makes when accomplishing a task.
There are two important differences between classical experiments and usability tests. Whereas the classical experimental method is designed to confirm or refute a hypothesis, the goal of usability tests is to obtain qualitative information on how to fix usability problems and how to improve the system. Another difference is the rigor with which the experiments need to be performed. It has been shown that even a series of quick focused tests starting as early as requirements elicitation is extremely helpful. Nielsen uses the term "discount usability engineering" to indicate that a few usability tests are better than none at all.

Question sources

Computer Science model exam 2 · Question 29

Question 83 | Q01260

Which one of the following is a Nonfunctional requirement?

A. Admin can upload photos

B. Admin can search book

C. Librarian can delete student

D. The system delivers the search items within 2 seconds

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: The system delivers the search items within 2 seconds

The two-second limit constrains how quickly the search service responds; the others specify permitted operations.

Why each choice is right or wrong

A - Incorrect: Uploading is a service. See PDF p. 65, 66.

B - Incorrect: Searching is a service. See PDF p. 65, 66.

C - Incorrect: Deletion is a service. See PDF p. 65, 66.

D - Correct: Quantitative response-time requirement. See PDF p. 65, 66.

Primary note proof

Study-note section SE10

> Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning. Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 65 | lines 19-23

> 3.2.1. Functional Requirements
> Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. For example, the following is an example of functional requirements for SatWatch, a watch that resets itself without user intervention:

Software Engineering module | PDF page 66 | lines 14-18

3.2.2. Non-Functional and Pseudo-Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Nonfunctional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands)
or accuracy (i.e., how precise are the system's numerical answers). The following are the

Question sources

Computer Science model exam 2 · Question 81

Question 84 | Q01261

Which of the following is false about functional requirement?

A. Requirements should complete

B. Requirement should be accurate as possible.

C. Each requirement should state a clear, independently verifiable obligation.

D. A requirement can be interpreted in many exclusive ways.

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: A requirement can be interpreted in many exclusive ways.

Requirements should accurately reflect stakeholder needs, cover the agreed scope and express clear, verifiable obligations. Unambiguity requires one intended interpretation; mutually exclusive readings violate that property. The revised third option states the useful single-obligation writing rule without claiming that duplicate requirements can never be consolidated. Thus the multiple-interpretation option is the single false quality claim, while completeness, accuracy and clarity remain valid goals.

Why each choice is right or wrong

A - Incorrect: Completeness requires coverage of the agreed behavior and exceptional cases.

B - Incorrect: Accuracy requires requirements to reflect the actual stakeholder needs.

C - Incorrect: A clear independently verifiable obligation supports precise specification.

D - Correct: Mutually exclusive readings violate the unambiguity requirement.

Primary note proof

Study-note section SE11

Correct requirements reflect actual stakeholder needs. Complete requirements address the agreed scope, normal and exceptional behavior. Consistency excludes contradictions. Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable

within constraints. Verifiability means objective evidence can check fulfillment. Replace “fast for most users” with a defined metric, workload and threshold.

Study-note section SE09

Write one clear obligation with an actor/trigger, action and observable result.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 68 | lines 18-26

system specification. Requirement validation involves checking if the specification is correct, complete, consistent, unambiguous, and realistic. A specification is correct if it represents the client’s view of the system (i.e., everything in the requirements model accurately represents an aspect of the system). It is complete if all possible scenarios through the system are described, including exceptional behavior (i.e., all aspects of the system are represented in the requirements
model). The system specification is consistent if it does not contradict itself. The system specification is unambiguous if exactly one system is defined (i.e., it is not possible to interpret the specification two or more different ways). Finally, it is realistic if the system can be implemented within constraints. These properties are illustrated with UML instance diagrams in

Question sources

Computer Science model exam 2 · Question 82

Question 85 | Q01272

Which of the following is not true about requirement elicitation?

A. Errors introduced during requirements elicitation are expensive to correct.

B. Define the purpose of the system by collaboration of the client and developers.

C. Requirements elicitation is less challenging than Analysis.

D. Requirements specification is the result of requirement elicitation.

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Requirements elicitation is less challenging than Analysis.

Elicitation requires stakeholder agreement and accurate problem understanding; analysis formalizes it. Elicitation is not inherently less challenging, and postponed decisions can be costly.

Why each choice is right or wrong

A - Incorrect: Late correction raises cost, consistent with identifying issues early. See PDF p. 62, 100.

B - Incorrect: Matches collaborative purpose definition. See PDF p. 62, 100.

C - Correct: The claimed easier-than-analysis ranking is unsupported and contrary to the described difficulty. See PDF p. 62, 100.

D - Incorrect: The system specification is its explicit output. See PDF p. 62, 100.

Primary note proof

Study-note section SE11

> Correct requirements reflect actual stakeholder needs. Complete requirements address the agreed scope, normal and exceptional behavior. Consistency excludes contradictions. Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable within constraints. Verifiability means objective evidence can check fulfillment. Replace “fast for most users” with a defined metric, workload and threshold.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 62 | lines 9-16

> Requirement’s elicitation focuses on describing the purpose of the system. The client, the developers, and the users identify a problem area and define a system that addresses the problem.
> Such a definition is called a system specification and serves as a contract between the client and the developers. The system specification is structured and formalized during analysis to produce an analysis model (see Figure 3.1). Both system specification and analysis model represent the same information. They differ only in the language and notation they use. The system specification
> is written in natural language, whereas the analysis model is usually expressed in a formal or semiformal notation.

Software Engineering module | PDF page 100 | lines 11-24

> Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users
> Figure 4.1 Products of requirements elicitation and analysis (UML activity diagram).
> Althoughtheanalysismodelmaynotbeunderstandabletotheusersandtheclient, it helps developers verify the system specification produced during requirements elicitation. There is a natural tendency for users and developers to postpone difficult decisions until later in the project. A decision may be difficult because of lack of domain knowledge, lack of technological knowledge, or simply because of disagreements among users and developers. Postponing decisions enables the
> project to move on smoothly and avoid confrontation with reality or peers. Unfortunately, difficult
> decisions will eventually need to be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation. Translating a system specification in to a formal or semi-formal model forces developers to identify and resolve difficult issues early in
> the development.

Question sources

Computer Science model exam 2 · Question 93

Question 86 | Q01512

Which named approach develops an experimental version specifically to explore unclear requirements or feasibility, and can therefore serve both as a development approach and an elicitation technique?

A. Spiral process model

B. Waterfall process model

C. Prototype process model

D. Agile process model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Prototype process model

Prototyping uses an early experimental representation or working version to discover requirements, usability needs or technical feasibility. It can be organized as a development approach and also used within elicitation. Spiral may contain a prototype to reduce risk, but risk-driven cycles define spiral itself. Waterfall names phased development, while Agile names a broader adaptive approach. The question asks the named experimentation technique that directly serves both purposes.

Why each choice is right or wrong

A - Incorrect: Spiral may employ prototypes but is itself defined by risk-driven cycles.

B - Incorrect: Waterfall is a phased lifecycle rather than the experimentation technique described.

C - Correct: Prototyping directly explores requirements or feasibility using an experimental version.

D - Incorrect: Agile is a broad adaptive approach and can use several elicitation techniques.

Primary note proof

Study-note section SE07

> A prototype explores requirements, usability or technical feasibility. A throwaway prototype is discarded after learning; an evolutionary prototype is deliberately refined toward production. A quick prototype may omit robustness, security or architecture, so do not deploy it unchanged without evaluation. Coding may occur during prototyping even if a diagram labels that activity construction rather than coding.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 26 | lines 6-9

> 2. Requirement's elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

Software Engineering module | PDF page 82 | lines 23-29

> • Prototype test. During this type of test, one or more users are presented with a piece of software that practically implements the system. A vertical prototype implements a complete use case through the system, and a horizontal prototype presents an interface for most use cases (without providing much or any functionality). The advantages of prototype tests are that they provide a realistic view of the system to the user and that prototypes can be instrumented to collect detailed data. The disadvantages of prototypes are that they are expensive to build and to modify.

Question sources

Mock exam 1 · Question 28

Question 87 | Q01513

What is the base for functional testing?

A. We have to refer to the system design document

B. We have to refer to the system requirement specification document

C. We have to refer to user manual if any exists

D. A and B

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: We have to refer to the system requirement specification document

Functional testing tests requirements from the RAD and, if available, the user manual. Of the options, the requirements specification is the primary basis; the user manual is supplementary.

Why each choice is right or wrong

A - Incorrect: Design goals support performance tests and architecture supports integration, rather than being the primary functional oracle. See PDF p. 144, 83, 130.

B - Correct: The requirement document defines what behavior to test. See PDF p. 144, 83, 130.

C - Incorrect: A manual can supplement functional requirements, but is not the primary complete basis. See PDF p. 144, 83, 130.

D - Incorrect: Adding design as an equal required functional basis overstates the cited rule. See PDF p. 144, 83, 130.

Primary note proof

Study-note section SE36

Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair. Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Software Engineering module | PDF page 83 | lines 12-25

3.4.4. Documenting Requirements Elicitation
The results of the requirements elicitation activity and the analysis activity are documented in the
Requirements Analysis Document (RAD). This document completely describes the system in terms of functional and nonfunctional requirements and serves as a contractual basis between the

client and the developers. The audience for the RAD includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design). The first part of the document, including use cases and nonfunctional requirements, is written during requirements
elicitation. The formalization of the specification in terms of object models is written during analysis.
The first section of the RAD is an Introduction. Its purpose is to provide a brief overview of the function of the system and the reasons for its development, its scope, and references to the development context (e.g., related problem statement, references to existing systems, feasibility studies). The introduction also includes the objectives and success criteria of the project.

Software Engineering module | PDF page 130 | lines 3-11

6.3. DOCUMENTING SYSTEM DESIGN
System design is documented in the System Design Document (SDD). It describes design goals set by the project, the subsystem decomposition (with UML class diagrams), the hardware/software mapping (with UML deployment diagrams), the data management, the access control, control flow mechanisms, and boundary conditions. The SDD is used to define interfaces between teams of developers and as reference when architecture-level decisions need to be revisited. The audience for the SDD includes the project management, the system architects (i.e., the developers who participate in the system design), and the developers who design and implement each subsystem. The following is an example template for a SDD:

Question sources

Mock exam 1 · Question 29

Question 88 | Q01514

Evaluate statements A-C about subsystem decomposition. Which is false? Select D if none is false.

A. In designing the desirable effect is low coupling and high cohesion in sub system decomposition

B. Cohesion refers to the dependency of classes with in subsystem

C. Coupling refers to the dependency between sub systems

D. None of statements A-C is false.

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: None of statements A-C is false.

Low coupling limits dependencies between subsystems, while high cohesion keeps related responsibilities together inside each subsystem. The first statement recommends this combination, the second identifies cohesion within the subsystem, and the third identifies coupling across subsystems. All three substantive statements are therefore true. The final alternative is explicitly the response for finding no false statement among those three, rather than a self-referential assertion about all choices.

Why each choice is right or wrong

A - Incorrect: Low coupling and high cohesion are the stated decomposition goals.

B - Incorrect: Cohesion concerns relatedness and dependencies among elements within a subsystem.

C - Incorrect: Coupling concerns dependencies between subsystems.

D - Correct: None of the substantive statements A-C is false, so this is the required response.

Primary note proof

Study-note section SE22

> Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module’s internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 120 | lines 6-12

> 6.1.3. Coupling and coherence
> Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
> is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Software Engineering module | PDF page 120 | lines 24-27

> Coherence is the strength of dependencies within a subsystem. If a subsystem contains many objects that are related to each other and perform similar tasks, its coherence is high. If a subsystem
> contains a number of unrelated objects, its coherence is low. A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence.

Question sources

Mock exam 1 · Question 30

Question 89 | Q01593

Which offered asset is inherently excluded from reuse in a software product line?

A. Test cases

B. Requirement

C. Architecture

D. None of A-C; all three can be reusable product-line assets.

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer D: None of A-C; all three can be reusable product-line assets.

Product-line assets can include requirements, architecture, components and tests. Consequently test cases, requirements and architecture are all potentially reusable, although each must be evaluated for fit in a particular product. None of A-C is inherently excluded by its asset type. Option D expresses this conclusion directly rather than contradicting the negative stem with an undefined all-choices assertion.

Why each choice is right or wrong

A - Incorrect: Test cases can be reusable product-line assets.

B - Incorrect: Requirements can be shared across related products.

C - Incorrect: Architecture can be a reusable product-line asset.

D - Correct: None of A-C is inherently excluded; all can be reused.

Primary note proof

Study-note section SE07

> RAD uses rapid construction, timeboxing, user participation and often reusable components; it requires available users and skilled designers/developers. Reuse-oriented development searches components, evaluates fit, may adjust requirements, designs around interfaces and builds missing parts. Product-line assets may include requirements, architecture, components and tests; all are potentially reusable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 24 | lines 20-29

> Advantages:
> 1. Reduced cost and risks
> 2. Faster delivery of the software (Accelerated development)
> Disadvantages:
> 1. Compromising requirements

2. Some control over the system evolution is lost as new versions of the reusable components are not under the control of the organization using them.
3. Increased maintenance costs
4. Not-invented-here syndrome
5. Lack of tool support

Question sources

Mock exam 3 · Question 25

Question 90 | Q01594

A software design model states exact withdraw preconditions and balance equations but omits physical implementation detail. Compare it with an informal mathematical sketch that leaves those constraints unspecified and a fully detailed engine drawing. Which statement is FALSE for these particular models?

A. This software model is less precise than the specified mathematical sketch.

B. This software model is more abstract than the fully detailed engine drawing.

C. A main role of this model is to represent the software designer’s ideas.

D. None of A-C is false.

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer A: This software model is less precise than the specified mathematical sketch.

The software model states exact preconditions and balance equations, whereas the informal mathematical sketch omits those constraints. It is therefore not less precise in the stated comparison, making A false. Omitting implementation detail makes it more abstract than the specified detailed engine drawing, and representing design ideas remains a legitimate purpose. These are scenario-specific judgments rather than universal rankings of all software and mathematical models.

Why each choice is right or wrong

A - Correct: Its exact constraints make the stated less-precise comparison false.

B - Incorrect: The software model omits implementation detail retained in the detailed drawing.

C - Incorrect: A design model can represent the designer’s ideas and decisions.

D - Incorrect: A is false, so the no-false-statements alternative is wrong.

Primary note proof

Study-note section SE43

Choose designs by their effect on stated goals. Fine-grained self-contained components, separation of producers/consumers and limited shared mutable data often improve maintainability. Trade-offs remain: interfaces can add latency, isolation can complicate transactions, and global state can simplify one operation while coupling many others. Prefer measurable quality scenarios to counting components alone.

Study-note section SE25

A contract states preconditions, postconditions and invariants. For withdraw(amount): require amount>0 and amount<=balance; ensure balance=oldBalance-amount; preserve balance>=0. Validation checks inputs; implementation enforces the transition. A complete feature needs both UI/API and persisted behavior, not only a diagram.

Study-note section SE23

A layer is a logical abstraction level; a tier is a deployment separation. A closed layered architecture permits calls only to the immediately lower layer; an open one permits approved skipping. For top-to-bottom A,B,C,D, A→B obeys closed layering, A→C skips, and lower-to-upper calls violate the simple dependency rule unless explicitly mediated by a defined mechanism.

Study-note section SE23

A structure is the elements/relations themselves; a view represents selected aspects.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 100 | lines 11-24

Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users
Figure 4.1 Products of requirements elicitation and analysis (UML activity diagram).
Althoughtheanalysismodelmaynotbeunderstandabletotheusersandtheclient, it helps developers verify the system specification produced during requirements elicitation. There is a natural tendency for users and developers to postpone difficult decisions until later in the project. A decision may be difficult because of lack of domain knowledge, lack of technological knowledge, or simply because of disagreements among users and developers. Postponing decisions enables the
project to move on smoothly and avoid confrontation with reality or peers. Unfortunately, difficult
decisions will eventually need to be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation. Translating a system specification in to a formal or semi-formal model forces developers to identify and resolve difficult issues early in
the development.

Software Engineering module | PDF page 18 | lines 13-20

System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
Software design involves identifying and describing the fundamental software system

abstractions
and their relationships. The goal of the design phase is to transform the requirements specified in
the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Question sources

Mock exam 3 · Question 26

Question 91 | Q01595

Which among the following designing models show sequence functional transformation done on inputs into outputs?

A. ER diagrams

B. DFD diagrams

C. Component diagrams

D. None of the choices are correct

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: Data-flow diagrams.

SE20 defines a DFD through data flows, stores and transformations from inputs to outputs. That matches the functional-transformation focus in the stem. An ER diagram focuses on domain data relationships, while a component diagram describes software components and dependencies. A DFD need not show detailed control timing merely because it depicts connected transformations.

Why each choice is right or wrong

A - Incorrect: ER diagrams describe entities and relationships rather than transformations of input data into outputs.

B - Correct: DFDs depict data flows and the processes that transform inputs into outputs.

C - Incorrect: Component diagrams describe component interfaces and dependencies rather than this data-transformation view.

D - Incorrect: DFD is listed and directly matches the functional-transformation description.

Primary note proof

Study-note section SE20

Choose a sequence diagram for message order between participants, a state machine for event-dependent state changes, and activity for workflow. Both sequence and state diagrams describe behavior over time. State machines are useful for objects with significant states, whether entity or control. A DFD shows data stores/flows and transformations from inputs to outputs; it is not a UML diagram.

Study-note section SE-S03

An entity-relationship diagram models entity types, their attributes and the relationships between them, commonly for data design. Its focus differs from a use-case diagram of external roles and goals, a class diagram of classes and their features, or a deployment diagram mapping artifacts to execution nodes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 126 | lines 7-16

Pipe and filter architecture
In the pipe and filter architecture (see Figure 6.13), subsystems process data received from a set of
inputs and send results to other subsystems via a set of outputs. The subsystems are called filters, and the associations between the subsystems are called pipes. Each filter only knows the content and the format of the data received on the input pipes, not the filters that produced them. Each filter is executed concurrently and synchronization is done via the pipes. The pipe and filter architecture is modifiable: Filters can be substituted for others or reconfigured to achieve a different purpose.
Figure 6.13 Pipe and filter architecture (UML class diagram). A Filter can have many inputs and outputs. A Pipe connects one of the outputs of a Filter to one of the inputs of another Filter.

Software Engineering module | PDF page 101 | lines 5-6

In UML, the functional model is represented with use case diagrams, the object model with class diagrams, and the dynamic model with statechart and sequence diagrams.

Question sources

Mock exam 3 · Question 27

Question 92 | Q01596

Which of the following is step in software development life cycle that tell us if developing a software makes sense for the current company situation or not.

A. Requirement analysis

B. Feasibility analysis

C. Designing phase

D. Requirement elicitation

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Feasibility analysis

Feasibility assesses technological practicality, budget and business cost-effectiveness before deciding to proceed with detailed analysis.

Why each choice is right or wrong

A - Incorrect: Further analysis follows a feasibility decision. See PDF p. 25, 26.

B - Correct: Direct viability/go-ahead decision. See PDF p. 25, 26.

C - Incorrect: Derives a solution after requirements are understood. See PDF p. 25, 26.

D - Incorrect: Collects stakeholder needs rather than deciding whether the investment is viable. See PDF p. 25, 26.

Primary note proof

Study-note section SE08

> Feasibility asks whether a proposal is worth doing and can be delivered within technical, economic, operational and schedule constraints. Examine existing limitations, organizational impacts, benefits, cost, available skills and integration risk. System analysis studies the problem and needed behavior in more detail; the activities inform each other.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 25 | lines 15-18

> There are four main activities in the requirements engineering process:
> 1. Feasibility study: Feasibility shows viability and practicality of a software, project or idea. An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study considers whether the proposed system will be cost-

Software Engineering module | PDF page 26 | lines 3-5

> effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.

Question sources

Mock exam 3 · Question 28

Question 93 | Q01597

Which choice gives the most complete supported comparison of object-oriented model continuity and transitions between different development models?

A. Object-oriented development can use a consistent vocabulary of objects and classes across analysis and design.

B. Reuse and consistent models in object-oriented development can reduce development effort and duration on suitable projects.

C. Moving between different representations in a traditional phased approach can add translation complexity and errors.

D. All statements A-C are correct.

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: All statements A-C are correct.

A consistent object vocabulary across development stages can reduce errors from translating between unrelated models. Reuse and continuity can also reduce effort on suitable projects, while transitions between different representations can add complexity. These are potential benefits and risks, not guarantees that every OO project is shorter or every traditional project is worse. With those qualifications, all three substantive statements are supported, making their aggregate the most complete answer.

Why each choice is right or wrong

A - Incorrect: True but incomplete alone: a consistent object/class vocabulary can connect analysis and design.

B - Incorrect: True but incomplete alone: reuse and model continuity can reduce effort, without guaranteeing it.

C - Incorrect: True but incomplete alone: translation between different representations can add complexity or errors.

D - Correct: All three qualified statements hold, so the aggregate is the complete selection.

Primary note proof

Study-note section SE01

> Software engineering applies systematic methods to development, operation and maintenance. A software product includes executable code and relevant requirements, designs, test assets, configuration and user documentation. A process defines activities, ordering, roles, products and entry/exit conditions. Its four broad activities are specification, design/implementation, validation and evolution.

Study-note section SE01

Procedural decomposition organizes functions; object oriented decomposition organizes objects with state and behavior. Either can use iterative or sequential development. OOP does not automatically shorten every project. Reuse, encapsulation and consistent models can reduce duplication and ease change; poor object design can still be complex. Functional programming emphasizes function evaluation and controlled side effects; event-driven programs react to events.

Study-note section SE-S02

An object-oriented approach can use a consistent vocabulary of objects, classes and relationships across analysis, design and implementation. Models are refined for different purposes rather than being identical artifacts. This continuity can reduce translation errors and effort; reusable classes can shorten suitable projects, but neither shorter duration nor good design is guaranteed.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 31 | lines 19-31

1.3.1. Why an object oriented
• Object oriented systems are
o Easier to adapt to changes
o More robust
o Easier to maintain
o Promote greater design and code reuse
o Creates modules of functionality
• REASONS FOR WORKING OF OBJECT ORIENTED SYSTEMS:
Higher level of abstraction
The object-oriented approach supports abstraction at the object level. Since objects encapsulate both data (attributes) and functions (methods), they work at a higher level of abstraction. The development can proceed at the object level and ignore the rest of the system for as long as necessary. This makes designing, coding, testing, and maintaining the system much simpler.

Software Engineering module | PDF page 32 | lines 22-27

1.3.2. Overview of the unified approach.
The unified modeling language (UML) is a set of notations and conventions used to describe and model an application. But, the UML does not specify a methodology or what steps to follow to develop an application; that would be the task of the UA. Figure 1.2 depicts the essence of the unified approach. The heart of the UA is Jacobson's use case. The use case represents a typical interaction between a user and a computer system to capture the users' goals and needs. The main

Software Engineering module | PDF page 12 | lines 3-17

Table 1.1. : Comparison of Traditional Approach vs Object oriented system
TRADITIONAL APPROACH OBJECT ORIENTED SYSTEM
DEVELOPMENT
Collection of procedures(functions) Combination of data and functionality
Focuses on function and procedures,
different styles and methodologies for each

step of process
Focuses on object, classes, modules that
can be easily replaced, modified and reused.
Moving from one phase to another phase
is complex.
Moving from one phase to another phase
is easier.
Increases duration of project decreases duration of project
Increases complexity Reduces complexity and redundancy

Software Engineering module | PDF page 32 | lines 3-10

The traditional approach to software development requires different styles and methodologies for
each step of the process. Moving from one phase to another requires a complex transition of perspective between models that almost can be in different worlds. This transition not only can slow the development process but also increases the size of the project and the chance for errors introduced in moving from one language to another. The object-oriented approach, on the other hand, essentially uses the same language to talk about analysis, design, programming, and database
design. This seamless approach reduces the level of complexity and redundancy and makes for clearer, more robust system development.

Question sources

Mock exam 3 · Question 29

Question 94 | Q01598

Which of the following process model is concerned with how changes are incorporated in software?

A. Configuration management process

B. Change management process

C. Inspection management process

D. Project management process

Hint

Separate artifact history and controlled changes from scheduling, collaboration benefits and automated execution.

Answer B: Change management process.

SE33 assigns handling change requests, impact assessment, decisions, implementation and validation to change management. Configuration management coordinates versions and baselines of affected artifacts, which supports that work but is a different focus. Inspection and general project management also contribute, yet B specifically names how proposed software changes are evaluated and incorporated.

Why each choice is right or wrong

A - Incorrect: Configuration management maintains controlled versions and baselines rather than primarily deciding and incorporating requested changes.

B - Correct: Change management evaluates, approves/prioritizes, implements and validates proposed changes.

C - Incorrect: Inspection evaluates artifacts rather than defining the complete change-handling process.

D - Incorrect: Project management is broader than the specific process for handling software change requests.

Primary note proof

Study-note section SE33

> Change management handles request, impact assessment, decision/prioritization, implementation and validation; configuration management keeps all affected versions consistent. Distributed Git gives each clone local history and supports independent branches followed by merging; it does not eliminate merge conflicts or approvals. Plan test configuration management early and apply it throughout testing, not at just one exclusive phase.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 5-10

> Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
> faults if change is controlled.

Question sources

Mock exam 3 · Question 30

Question 95 | Q01599

After a maintenance change request has been received, which is the earliest listed technical activity before release planning and implementation?

A. Release planning

B. Change management

C. Impact analysis

D. Testing

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer C: Impact analysis

A typical evolution sequence is change request, impact analysis, release planning, implementation/testing and release. Once the request has arrived, impact analysis is the earliest listed technical step, assessing consequences before scheduling and implementation. Change management names the broader handling process rather than that next step. Release planning and testing occur later, so neither starts the specified post-request technical sequence.

Why each choice is right or wrong

A - Incorrect: Release planning follows impact analysis.

B - Incorrect: Change management is the overarching process.

C - Correct: Impact analysis is the first listed technical activity after the request.

D - Incorrect: Testing follows analysis, planning and implementation.

Primary note proof

Study-note section SE40

> A typical evolution sequence is change request→impact analysis→release planning→implementation/testing→release. Among a list lacking the request step, impact analysis may be the earliest listed technical activity. Major restructuring is reengineering; if the architecture itself changes, architectural transformation is the more specific description. Markets, technology, stakeholders and funding can all change scope or priorities.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 19 | lines 24-29

> 5. Operation and maintenance: Normally (although not necessarily), this is the longest life cycle phase. Maintenance of a typical software product requires much more than the effort necessary to
> develop the product itself. The system is installed and put into practical use. Maintenance involves

correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system ‘s services as new requirements are discovered.

Software Engineering module | PDF page 41 | lines 16-17

✓ Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.

Question sources

Mock exam 3 · Question 31

Question 96 | Q01600

A takes 5 days. B takes 3 days and C takes 4 days; both depend only on A and may run in parallel with sufficient resources. The project finishes when all three complete. What is its earliest duration?

A. 3 days

B. 4 days

C. 5 days

D. 9 days

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer D: 9 days

A completes at day five. B and C then start together, finishing at days eight and nine respectively. The project must wait for both required tasks, so its earliest finish is 5+max(3,4)=9 days. Adding all durations would ignore the permitted parallelism, while using eight days would count only the shorter A-B path and leave C incomplete.

Why each choice is right or wrong

A - Incorrect: Three counts B alone and omits its predecessor.

B - Incorrect: Four counts C alone and omits A.

C - Incorrect: Five stops after A, before both successors.

D - Correct: A followed by the longer parallel branch takes nine days.

Primary note proof

Study-note section SE29

If A takes 5 days and B=3,C=4 both follow A, project duration is 5+max(3,4)=9 days. B finishes at 8, but the project waits for C at 9. If paths B5+C6 and D8+E3 share a zero-duration start/end, both total 11; there are two critical paths, not one.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 3-12

4.6. Scheduling, Organization and Team Structures
Project-task scheduling is an important project planning activity. It involves deciding which tasks would
be taken up when. In order to schedule the project activities, a software project manager needs to do
the following:
• Identify all the tasks needed to complete the project and Break down large tasks into small activities.
• Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
• Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Mock exam 3 · Question 32

Question 97 | Q01601

A project starts at day 0 and must finish at day 9. It consists of exactly two parallel paths, A → B and C → D. A and C have no predecessors, B depends only on A, D depends only on C, and the project finishes when both B and D are complete. Activity durations and early times are: A: duration 4 days, ES=0, EF=4; B: duration 5 days, ES=4, EF=9; C: duration 3 days, ES=0, EF=3; D: duration 6 days, ES=3, EF=9. What is the total slack (float) of Activity C?

A. 0 days

B. 1 day

C. 3 days

D. 6 days

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer A: 0 days.

Each complete path takes 9 days: A→B is 4+5 and C→D is 3+6. Working backward from day 9, D must start by day 3. Therefore C must finish by day 3 and start by day 0. C has LS=0 and ES=0, so total slack = LS−ES = 0 days (equivalently LF−EF = 3−3). A is correct. Delaying C would delay its successor D and the required project finish.

Why each choice is right or wrong

A - Correct: C has latest start 0 and earliest start 0, giving zero total slack.

B - Incorrect: A one-day delay would make the C→D path finish on day 10, beyond the day-9 finish.

C - Incorrect: C's three-day duration is not its slack; a three-day delay would finish this path on day 12.

D - Incorrect: D's six-day duration is not C's slack; a six-day delay to C would finish the path on day 15.

Primary note proof

Study-note section SE30

> Early start/finish values alone do not determine slack without successor relationships and the project completion constraint. If C3→D6 is a complete path parallel to A4→B5 and both finish at day9, C has zero float. That network cannot be silently inferred from four unrelated ES/EF rows.

Study-note section SE30

> Traditional CPM uses point durations; PERT models uncertainty with optimistic a, most-likely m and pessimistic b. The classic estimate is TE=(a+4m+b)/6 and variance=((b-a)/6)^2. For 3,6,15 days, TE=42/6=7 days and variance=4 days². Do not replace the weighted estimate by a simple average. These formulas are approximations, not guaranteed actual completion times.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 3-12

> 4.6. Scheduling, Organization and Team Structures
> Project-task scheduling is an important project planning activity. It involves deciding which tasks would
> be taken up when. In order to schedule the project activities, a software project manager needs to do
> the following:
> • Identify all the tasks needed to complete the project and Break down large tasks into small activities.
> • Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
> • Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Mock exam 3 · Question 33

Question 98 | Q01602

PERT considers which of the following time estimates for each activity?

A. Ideal time only

B. Worst-case time only

C. Three-point estimates (optimistic, most likely, pessimistic)

D. Average of historical data (if available) statement.

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer C: Optimistic, most likely and pessimistic estimates.

SE30 defines the classic PERT inputs as optimistic a, most-likely m and pessimistic b. These feed the weighted expected time (a+4m+b)/6 and an uncertainty estimate. Neither a single ideal or worst-case time nor an unqualified historical average supplies the three-point representation of uncertainty described by PERT.

Why each choice is right or wrong

A - Incorrect: An ideal estimate alone omits the most-likely and pessimistic values needed by classic PERT.

B - Incorrect: A worst-case estimate alone omits the optimistic and most-likely possibilities.

C - Correct: Classic PERT uses optimistic, most-likely and pessimistic duration estimates.

D - Incorrect: Historical data may inform estimates, but an average alone is not the PERT three-point input model.

Primary note proof

Study-note section SE30

> Traditional CPM uses point durations; PERT models uncertainty with optimistic a, most-likely m and pessimistic b. The classic estimate is TE=(a+4m+b)/6 and variance=((b-a)/6)^2. For 3,6,15 days, TE=42/6=7 days and variance=4 days^2. Do not replace the weighted estimate by a simple average. These formulas are approximations, not guaranteed actual completion times.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 3-12

> 4.6. Scheduling, Organization and Team Structures
> Project-task scheduling is an important project planning activity. It involves deciding which tasks

would
be taken up when. In order to schedule the project activities, a software project manager needs to do
the following:
• Identify all the tasks needed to complete the project and Break down large tasks into small activities.
• Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
• Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Mock exam 3 · Question 34

Question 99 | Q01603

_____diagram is not supported by UML considering Data-driven modeling.

A. Activity

B. Data Flow Diagram (DFD)

C. State Chart

D. Component

Hint

Match the information to be represented with each model’s purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: Data Flow Diagram (DFD)

The traditional nine UML diagram types include activity, statechart and component diagrams; DFD is not a UML diagram type. Thus DFD is the offered non-UML notation.

Why each choice is right or wrong

A - Incorrect: Listed UML activity diagram. See PDF p. 101, 56, 55.

B - Correct: Not included among the module's nine UML diagrams on PDF p.47. See PDF p. 101, 56, 55.

C - Incorrect: Listed UML statechart diagram. See PDF p. 101, 56, 55.

D - Incorrect: Listed UML component diagram on PDF p.47. See PDF p. 101, 56, 55.

Primary note proof

Study-note section SE20

Choose a sequence diagram for message order between participants, a state machine for event-dependent state changes, and activity for workflow. Both sequence and state diagrams describe behavior over time. State machines are useful for objects with significant states,

whether entity or control. A DFD shows data stores/flows and transformations from inputs to outputs; it is not a UML diagram.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 101 | lines 5-6

In UML, the functional model is represented with use case diagrams, the object model with class diagrams, and the dynamic model with statechart and sequence diagrams.

Software Engineering module | PDF page 56 | lines 18-24

2.3.4. Activity diagrams
The outgoing transitions are triggered by the completion of an action associated with the state. This is called an action state. By convention, the name of a state denotes a condition, whereas the name of an action state denotes an action. Activity diagrams are statechart diagrams whose states
are action states. is an activity diagram corresponding to the state diagram. An alternate and equivalent view of activity diagrams is to interpret action states as control flow between activities
and transitions; that is, the arrows are interpreted as sequential constraints between activities.

Software Engineering module | PDF page 55 | lines 4-9

2.3.3. State chart Diagram
A UML statechart is a notation for describing the sequence of states an object goes through in response to external events. Statecharts are extensions of the finite state machine model. On the one hand, statecharts provide notation for nesting states and state machines (i.e., a state can be described by a state machine). On the other hand, statecharts provide notation for binding transitions with message sends and conditions on objects.

Software Engineering module | PDF page 47 | lines 15-27

UML includes the following nine diagrams and the details are described in the following chapters.
• Class diagram
• Object diagram
• Use case diagram
• Sequence diagram
• Collaboration
diagram
• Activity diagram
• State chart diagram
• Deployment
diagram
• Component
diagram

Question sources

Mock exam 3 · Question 35

Question 100 | Q01604

What defining intention distinguishes a throwaway prototype from an evolutionary prototype?

A. It involves successive steps

B. It involves just one task

C. The prototype is built with the idea that it will eventually be converted into final system

D. It is intended to be discarded after learning, rather than refined into the production system.

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer D: It is intended to be discarded after learning, rather than refined into the production system.

A throwaway prototype is built to learn about requirements, usability or feasibility and is discarded after serving that purpose. An evolutionary prototype is deliberately refined toward production. Either may involve successive steps, multiple tasks and varying development time, so those features do not define the difference. The artifact's intended future, rather than how quickly it was built, distinguishes the approaches.

Why each choice is right or wrong

A - Incorrect: Either kind may involve successive steps.

B - Incorrect: Either kind can explore more than one task.

C - Incorrect: Intended refinement into the final system describes evolutionary prototyping.

D - Correct: Discard after learning is the defining throwaway intention.

Primary note proof

Study-note section SE07

> A prototype explores requirements, usability or technical feasibility. A throwaway prototype is discarded after learning; an evolutionary prototype is deliberately refined toward production. A quick prototype may omit robustness, security or architecture, so do not deploy it unchanged without evaluation. Coding may occur during prototyping even if a diagram labels that activity construction rather than coding.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 82 | lines 23-29

> • Prototype test. During this type of test, one or more users are presented with a piece of software that practically implements the system. A vertical prototype implements a complete use case through the system, and a horizontal prototype presents an interface for most use cases (without providing much or any functionality). The advantages of prototype

tests are that they provide a realistic view of the system to the user and that prototypes can be instrumented to collect detailed data. The disadvantages of prototypes are that they are expensive to build and to modify.

Software Engineering module | PDF page 26 | lines 6-9

2. Requirement's elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

Question sources

Mock exam 3 · Question 36

Question 101 | Q01605

Which factors can cause requirements, scope or priorities to evolve during a project? Select all that apply.

A. Changing needs of stakeholders

B. Advances in technology

C. Market dynamics

D. Lack of project funding

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer A, B, C, D: Changing needs of stakeholders; Advances in technology; Market dynamics; Lack of project funding

Stakeholder needs can change the services required, technology can introduce new constraints or opportunities, and market dynamics can alter business priorities. A funding shortfall can also reduce scope or force reprioritization. The note explicitly includes all four sources of change. Their effects depend on the project, but each can cause evolution, so none is an unconditional exception.

Why each choice is right or wrong

A - Correct: Changed stakeholder needs can revise required behavior.

B - Correct: Technology changes can alter constraints or opportunities.

C - Correct: Market changes can alter goals and priorities.

D - Correct: Funding limits can reduce scope or reorder requirements.

Primary note proof

Study-note section SE11

Trace each requirement backward to its source/rationale and forward to design, code and tests. Record dependencies among requirements. A traceability matrix links these artifacts and helps impact analysis; it is not just a cost/schedule ledger. Stakeholders must negotiate conflicting requirements rather than independently approving incompatible demands. Scope, understanding and volatility are distinct elicitation difficulties.

Study-note section SE40

Corrective maintenance fixes faults; adaptive maintenance responds to environment changes; perfective maintenance improves useful capability/performance; preventive maintenance reduces future risk, such as restructuring fragile code. Motive determines category; refactoring can support preventive or perfective work. Maintenance often lasts years and includes understanding legacy code and regression effort, so it may dominate lifecycle cost.

Study-note section SE40

Markets, technology, stakeholders and funding can all change scope or priorities.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 41 | lines 16-17

✓ Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.

Software Engineering module | PDF page 96 | lines 12-21

4.7. Risk Management
• Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
• A risk is a probability that some adverse circumstance will occur
- Project risks affect schedule or resources;
- Product risks affect the quality or performance of the software being developed;
- Business risks affect the organisation developing or procuring the software.
Risk Management Process
- Risk identification- Identify project, product and business risks;
- Risk analysis -Assess the likelihood and consequences of these risks;

Question sources

Mock exam 3 · Question 37

Question 102 | Q01606

Used when a large amount of testing is needed and the changes, while small, can affect many parts of the system

A. Preventive testing

B. Unit Testing

C. Regression Testing

D. Acceptance Testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: Regression testing.

The concern is that small changes may disturb other previously working parts of the software. Regression testing means checking unintended effects after change, rather than only confirming the defect. Such testing can span multiple levels and a large scope; the question describes its purpose, not the size of a unit or an acceptance decision.

Why each choice is right or wrong

A - Incorrect: Preventive testing is not the defined technique for detecting unintended effects of a specific software change.

B - Incorrect: Unit testing isolates a small component and does not by itself define the cross-system change-impact purpose.

C - Correct: Regression tests check whether changes unintentionally damage previously working behavior elsewhere.

D - Incorrect: Acceptance testing supports a user acceptance decision rather than specifically checking unintended change effects.

Primary note proof

Study-note section SE36

> Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair. Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 5-10

> Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all

developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
faults if change is controlled.

Software Engineering module | PDF page 147 | lines 14-18

• Unit testing, which finds faults by isolating an individual component using test stubs and drivers and by exercising the component using a test case
• Integration testing, which finds faults by integrating several components together
• System testing, which focuses on the complete system, its functional and nonfunctional requirements, and its target environment

Software Engineering module | PDF page 147 | lines 3-8

• A test stub is a partial implementation of components on which the tested component depends. A test driver is a partial implementation of a component that depends on the tested component. Test stubs and drivers enable components to be isolated from the rest of the system for testing.
• A correction is a change to a component. The purpose of a correction is to repair a fault. Note that a correction can introduce new faults.

Question sources

Mock exam 3 · Question 38

Question 103 | Q01607

The Decision table testing is a ----------?

A. White box Test Design Technique

B. Black Box Test Design Technique

C. Experience-based Test Design Technique

D. Grey Box Test Design Technique

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: Black-box, specification-based testing.

SE37 derives decision-table cases from combinations of specified conditions and resulting actions. This is specification-based rather than code-structure-based testing. It therefore belongs to black-box techniques. Experience or partial internal knowledge may help a tester, but they do not define the decision-table method or change its condition/action test basis.

Why each choice is right or wrong

A - Incorrect: White-box techniques derive tests from internal implementation structure rather than specified condition/action rules.

B - Correct: Decision tables derive cases from specified conditions and expected actions, making them black-box.

C - Incorrect: The technique has an explicit specification-based structure rather than being defined by tester experience.

D - Incorrect: Partial knowledge of internals is not required to derive decision-table test cases.

Primary note proof

Study-note section SE37

> A decision table lists condition combinations and resulting actions. For validCard and enoughFunds, enumerate TT,TF,FT,FF; specify dispense only for TT and the proper rejection for others. Cover feasible rules rather than blindly using every impossible combination. These are specification-based techniques. Test requirements can also define state-transition cases, such as rejecting dispatch from an already closed incident.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 6-10

> Equivalence testing is a black box testing technique that minimizes the number of test cases. The possible inputs are partitioned into equivalence classes, and a test case is selected for each class. The assumption of equivalence testing is that systems usually behave in similar ways for all members of a class. To test the behavior associated with an equivalence class, we only need to test
> one member of the class. Equivalence testing consists of two steps: identification of the

Software Engineering module | PDF page 149 | lines 24-29

> Path testing
> Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path testing is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.
> The starting point for path testing is the flow graph. A flow graph consists of nodes representing

Question sources

Mock exam 3 · Question 39

Question 104 | Q01608

Which of the following testing is related to the boundary value analysis?

A. White & black box testing

B. White-box testing

C. Black box testing

D. None of the above

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: Black box testing

Boundary testing is a special case of equivalence testing, a black-box testing technique.

Why each choice is right or wrong

A - Incorrect: The cited classification does not require both white and black knowledge. See PDF p. 149, 149.

B - Incorrect: Internal path knowledge is not required by input-class edges. See PDF p. 149, 149.

C - Correct: Follows by boundary being a special case of black-box equivalence testing. See PDF p. 149, 149.

D - Incorrect: The black-box option is supported. See PDF p. 149, 149.

Primary note proof

Study-note section SE37

> Equivalence partitioning groups inputs expected to behave alike, then samples valid and invalid groups. For allowed age18..60, useful partitions are below18,18..60,above60. Boundary tests target edges and neighbors, such as 17,18,19,59,60,61, with the exact set chosen under the stated coverage scheme.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 18-23

> Boundary testing
> Boundary testing is a special case of equivalence testing and focuses on the conditions at the boundary of the equivalence classes. Rather than selecting any element in the equivalence class, boundary testing requires that the elements be selected from the “edges” of the equivalence class.
> The assumption behind boundary testing is that developers often overlook special cases at the boundary of the equivalence classes (e.g., 0, empty strings, year 2000).

Software Engineering module | PDF page 149 | lines 6-10

Equivalence testing is a black box testing technique that minimizes the number of test cases. The possible inputs are partitioned into equivalence classes, and a test case is selected for each class. The assumption of equivalence testing is that systems usually behave in similar ways for all members of a class. To test the behavior associated with an equivalence class, we only need to test
one member of the class. Equivalence testing consists of two steps: identification of the

Question sources

Mock exam 3 · Question 40

Question 105 | Q01636

During initial design, a component with three unrelated responsibilities is split into three cohesive components. Which concept most specifically names the resulting decomposition into responsibility-based modules, rather than a general restructuring activity?

A. Refactoring

B. Modularization

C. Information hiding

D. Refinement

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Modularization

Separating unrelated responsibilities into cohesive components is modularization: the design is divided into modules with focused purposes. Refactoring can describe behavior-preserving restructuring of an existing solution, but that broader activity is not the specific decomposition concept requested. Information hiding concerns concealed decisions behind interfaces, while refinement is a general elaboration concept. The stated split primarily establishes better module responsibility boundaries.

Why each choice is right or wrong

A - Incorrect: Refactoring names behavior-preserving restructuring rather than the specific module decomposition principle.

B - Correct: Modularization separates responsibilities into cohesive components.

C - Incorrect: No particular hidden implementation decision defines the stated split.

D - Incorrect: Refinement is broader than the specified module-boundary decomposition.

Primary note proof

Study-note section SE22

Logical cohesion groups similar-category operations, often selected by a flag; temporal cohesion groups operations done at the same time; functional cohesion focuses on one task. Splitting an unrelated three-purpose component improves modularity. If behavior is preserved, the restructuring is refactoring; the words are not mutually exclusive answers. Smaller modules can aid understanding, independent work, reuse and maintenance, but too many interfaces also create complexity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 120 | lines 24-27

Coherence is the strength of dependencies within a subsystem. If a subsystem contains many objects that are related to each other and perform similar tasks, its coherence is high. If a subsystem
contains a number of unrelated objects, its coherence is low. A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence.

Software Engineering module | PDF page 133 | lines 3-15

transformation on subsets of a particular model. Typical activities include transforming n-ary associations into binary associations, implementing binary associations as references, merging two similar classes from two different subsystems into a single class, collapsing classes with no significant behavior into attributes, splitting complex classes into simpler ones, rearranging classes and operations to increase the inheritance and packaging. During restructuring, we address design goals such as maintainability, readability, and understandability of the system model.
• Optimization. Optimization activities address performance requirements of the system model. This includes changing algorithms to respond to speed or memory requirements, reducing multiplicities in associations to speed up queries, adding redundant associations for efficiency, rearranging execution orders, adding derived attributes to improve the access time to objects and opening up the architecture, that is, adding access to lower layers because of performance requirements.

Question sources

Mock exam 2 · Question 25

Question 106 | Q01637

Which design approach characteristically starts with available reusable components and assembles them into a larger system, rather than starting from high-level decomposition?

A. Both can be used

B. Top down

C. Bottom up

D. None of the choices are correct

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Bottom up

Bottom-up design begins with existing components and assembles them into a larger solution. Top-down design instead starts with high-level structure and refines it, although it can still reuse components later. A hybrid combines the two directions, but the question asks which approach characteristically begins with the parts. Reuse by itself would not distinguish them; the explicit starting direction does.

Why each choice is right or wrong

A - Incorrect: A hybrid may use both directions but does not name the specifically described starting approach.

B - Incorrect: Top-down begins with high-level structure.

C - Correct: Bottom-up assembles existing components into the larger system.

D - Incorrect: Bottom-up directly matches, so none is false.

Primary note proof

Study-note section SE25

> Use abstraction and decomposition to manage complexity; preserve consistency, traceability and clear responsibilities. Top-down design refines high-level structure; bottom-up design assembles existing components. A hybrid combines both. SOLID includes Single Responsibility, Open-Closed, Liskov substitution, Interface Segregation and Dependency Inversion. Depend on abstractions; add compatible extensions; avoid forcing unused operations. DRY limits duplicated knowledge, KISS avoids unnecessary complexity, and YAGNI avoids speculative work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 24 | lines 20-29

> Advantages:
> 1. Reduced cost and risks
> 2. Faster delivery of the software (Accelerated development)
> Disadvantages:
> 1. Compromising requirements
> 2. Some control over the system evolution is lost as new versions of the reusable components are not under the control of the organization using them.
> 3. Increased maintenance costs
> 4. Not-invented-here syndrome
> 5. Lack of tool support

Software Engineering module | PDF page 18 | lines 13-20

System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
Software design involves identifying and describing the fundamental software system abstractions
and their relationships. The goal of the design phase is to transform the requirements specified in
the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Question sources

Mock exam 2 · Question 26

Question 107 | Q01638

Processes and threads are element types we find in _____ software structure.

A. Service software structure

B. Use software structure

C. Layer software structure

D. Concurrent software structure

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: Concurrent software structure.

SE23 explicitly puts processes and threads in concurrency views because they represent independently scheduled execution. Layer structure instead describes logical abstraction levels, while uses and service relations describe different dependencies or interfaces. Thus the concurrent structure is the offered category that directly matches these runtime execution elements.

Why each choice is right or wrong

A - Incorrect: Service structure emphasizes provided behavior or interfaces rather than processes and threads as execution elements.

B - Incorrect: Uses structure emphasizes dependencies rather than the concurrency entities named in the stem.

C - Incorrect: Layers organize logical abstraction levels rather than identifying independently scheduled execution units.

D - Correct: Concurrent structure models execution elements such as processes and threads.

Primary note proof

Study-note section SE23

A structure is the elements/relations themselves; a view represents selected aspects. Module structures show code organization, component-and-connector structures runtime interaction, and allocation structures map software to hardware, teams or environment. Processes/threads belong in concurrency views. Evaluate architectures against quality attributes and scenarios; processor allocation, decomposition and style selection can all be architectural concerns.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 121 | lines 6-15

6.1.4. Software architecture
As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
In this section, we describe a few sample architectures that can be used for different systems. This
is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
representative examples and refer the reader to the literature for more details.

Question sources

Mock exam 2 · Question 27

Question 108 | Q01639

Which statements correctly describe verification and validation? Select all that apply.

A. Verification consists of checking of documents/files and is performed by human where as validation consists of execution of program and is performed by computer.

B. All choices are correct

C. Validation assesses whether the built software meets intended stakeholder needs.

D. Verification refers to the set of activities that ensure software correctly implements the specific function.

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer C, D: Validation assesses whether the built software meets intended stakeholder needs.; Verification refers to the set of activities that ensure software correctly implements the specific function.

Verification checks whether specified functions and obligations have been implemented correctly. Validation checks whether the built product meets intended stakeholder needs. Both can use reviews, analysis and execution, so the claimed human-only versus computer-only split is false. Traceability can support these activities but does not alone establish satisfaction. If C states that validation checks whether software meets intended stakeholder needs, C and D are correct.

Why each choice is right or wrong

A - Incorrect: Both activities can use human reviews and execution; the exclusive split is false.

B - Incorrect: A is false, so all choices are not correct.

C - Correct: Validation assesses whether the product meets intended stakeholder needs.

D - Correct: Verification assesses correct implementation of specified obligations.

Primary note proof

Study-note section SE34

> Quality assurance establishes and checks processes/standards; quality control evaluates work products. An SQA plan identifies standards, responsibilities, evaluations, reviews, audits and assurance records. Quality of conformance means following the design/specification; a product can conform to a poor specification. Verification asks whether specified obligations are implemented correctly; validation asks whether the product meets intended stakeholder needs. Both can use reviews, analysis or execution, so “human-only versus computer-only” is false.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 11-23

> Verification attempts to find faults before any execution of the system. Verification is possible in specific cases, such as the verification of a small operating system kernel. Verification, however, has limits. It is not in a mature enough state that it can be applied to assure the quality of large complex systems. Moreover, it assumes that the requirements are correct, which is seldom the case: Assume we need to verify an operation and we know the pre- and postconditions for the operation. If we can verify that the operation provides a transformation such that, before the execution of the operation the precondition is true, and after the execution the postcondition holds,
> then the operation is correctly implemented; that is, it is verified. However, sometimes we forget important parts of the pre- or postcondition. For example, the postconditions might not state that the train should stay on the ground while moving from track 1 to track 2. Finally, verification addresses only algorithmic faults: It cannot deal with mechanical faults in the virtual machine. For
> verification, we must assume that the virtual machine on which the system is executing is correct
> and does not change during the proof.

Question sources

Mock exam 2 · Question 28

Question 109 | Q01640

Which architectural style is more suitable for if a system has to have successive transformation of streams of data?

A. Pipe and filter

B. Layered architecture

C. Client server architecture

D. All choices are correct

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer A: Pipe and filter

Pipe-and-filter architecture explicitly applies transformations to streams without user intervention.

Why each choice is right or wrong

A - Correct: Direct match to successive stream transformations. See PDF p. 126, 127, 124.

B - Incorrect: Layers organize levels, but the module does not identify them as the stream transformation style. See PDF p. 126, 127, 124.

C - Incorrect: Client/server organizes requests/services, not the requested transformation pipeline. See PDF p. 126, 127, 124.

D - Incorrect: Client/server is not equally established for the specific stream criterion. See PDF p. 126, 127, 124.

Primary note proof

Study-note section SE24

> Repository style shares central data, simplifying common access but creating bottleneck/schema dependencies. MVC separates domain model, display view and interaction controller. Client-server separates requesters from service providers; peer-to-peer permits both roles. Pipe-and-filter applies successive transformations to data streams. Three-tier separates presentation, application logic and storage; some four-tier variants split presentation client/server.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 126 | lines 7-16

Pipe and filter architecture
In the pipe and filter architecture (see Figure 6.13), subsystems process data received from a set of
inputs and send results to other subsystems via a set of outputs. The subsystems are called filters, and the associations between the subsystems are called pipes. Each filter only knows the content and the format of the data received on the input pipes, not the filters that produced them. Each filter is executed concurrently and synchronization is done via the pipes. The pipe and filter architecture is modifiable: Filters can be substituted for others or reconfigured to achieve a different purpose.
Figure 6.13 Pipe and filter architecture (UML class diagram). A Filter can have many inputs and outputs. A Pipe connects one of the outputs of a Filter to one of the inputs of another Filter.

Software Engineering module | PDF page 127 | lines 5-8

Pipe and filter architectures are suited for systems that apply transformations to streams of data without intervention by users. They are not suited for systems that require more-complex interactions between components, such as an information management system or an interactive system.

Software Engineering module | PDF page 124 | lines 8-16

Client/server architecture
In the client/server architecture (Figure 6.9), a subsystem, the server, provides servicesto instances
of other subsystems called the clients, which are responsible for interacting with theuser. The request for a service is usually done via a remote procedure call mechanism or acommon object broker (e.g., CORBA or Java RMI). Control flow in the clients and the servers isindependent except for synchronization to manage requests or to receive results.
Figure 6.9 Client/server architecture (UML class diagram).
Clients request services from one or more servers. The server has no knowledge of the client. The client/server architecture is a generalization of the repository architecture.

Question sources

Mock exam 2 · Question 29

Question 110 | Q01641

Which of the following is quality measure for software?

A. Reliability

B. Low cost

C. Productivity

D. Low development time

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer A: Reliability

Reliability is a software quality attribute characterized by behavior conformance and failure probability.

Why each choice is right or wrong

A - Correct: Explicit quality metric. See PDF p. 91, 140.

B - Incorrect: A project cost consideration rather than the offered software-behavior quality metric. See PDF p. 91, 140.

C - Incorrect: An output/effort process measure, not the specific product quality attribute. See PDF p. 91, 140.

D - Incorrect: A schedule measure rather than product reliability. See PDF p. 91, 140.

Primary note proof

Study-note section SE34

> A fault/defect is an underlying problem; an error may be an incorrect internal state; a failure is an externally observed departure from required service. Reliability is a product quality attribute; low cost or short development time are project outcomes. Tests increase confidence and reveal defects but cannot generally guarantee correctness over every possible input.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 91 | lines 37-42

> • Software Metrics can be used to indicate the basic attributes of the software. The indicative metrics, not exhaustive, are
> • Size - Line of code ,Function point count
> • Quality- Reliability, Accuracy, Dependability, and these are measured through some measures like errors, failures and number of runs between failures ,etc.
> • Execution Time -process time of execution of a critical process and many more.

Software Engineering module | PDF page 140 | lines 11-18

> Reliability is a measure of success with which the observed behavior of a system conforms to the specification of its behavior. Software reliability is the probability that a software system will not cause the failure of the system for a specified time under specified conditions. Failure is any deviation of the observed behavior from the specified behavior. An error means the system is in a state such that further processing by the system will lead to a failure, which causes the system to
> deviate from its intended behavior. A fault, also called defect or bug, is the mechanical or algorithmic cause of an error. The goal of testing is to maximize the number of discovered faults, which then allows developers to correct them and increase the reliability of the system.

Question sources

Mock exam 2 · Question 30

Question 111 | Q01642

A project manager is using PERT to estimate the duration of a project activity. The optimistic time (a) is 3 days, the most likely time (m) is 6 days, and the pessimistic time (b) is 15 days. What is the expected time (TE) for the activity?

A. 8 days

B. 10 days

C. 7 days

D. 9 days

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer C: 7 days.

Use TE=(a+4m+b)/6 from SE30 with a=3, m=6 and b=15. The numerator is 3+24+15=42, so TE=42/6=7 days. A simple average would give 8, but that discards the fourfold weighting of the most-likely estimate. The expected duration is an estimate, not a guaranteed actual completion time.

Why each choice is right or wrong

A - Incorrect: Eight is the simple mean of 3, 6 and 15, not the weighted PERT estimate.

B - Incorrect: Ten does not result from substituting the three given values into the PERT formula.

C - Correct: The weighted sum is 42 and dividing by six gives 7 days.

D - Incorrect: Nine does not equal the PERT weighted expected time for these estimates.

Primary note proof

Study-note section SE30

> Traditional CPM uses point durations; PERT models uncertainty with optimistic a, most-likely m and pessimistic b. The classic estimate is TE=(a+4m+b)/6 and variance=((b-a)/6)^2. For 3,6,15 days, TE=42/6=7 days and variance=4 days². Do not replace the weighted estimate by a simple average. These formulas are approximations, not guaranteed actual completion times.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 3-12

> 4.6. Scheduling, Organization and Team Structures
> Project-task scheduling is an important project planning activity. It involves deciding which tasks would
> be taken up when. In order to schedule the project activities, a software project manager needs to do
> the following:

• Identify all the tasks needed to complete the project and Break down large tasks into small activities.
• Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
• Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Mock exam 2 · Question 31

Question 112 | Q01643

Which of the following is NOT a characteristic of a critical path in a CPM network diagram for software development?

A. Any delay in tasks on this path delays the entire project.

B. It identifies tasks with strict dependencies.

C. It represents the longest sequence of tasks.

D. It has the most slack (buffer time) compared to other paths.

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer D: It has the most slack (buffer time) compared to other paths.

The critical path is the chain determining project duration. The maximum-slack claim contradicts this: freely postponable tasks would not determine the earliest completion under a fixed network.

Why each choice is right or wrong

A - Incorrect: Delaying a controlling chain delays completion. See PDF p. 95.

B - Incorrect: Dependency order constrains the chain, though noncritical tasks can also have strict dependencies. See PDF p. 95.

C - Incorrect: The duration-determining chain is the longest required chain in the fixed-duration network. See PDF p. 95.

D - Correct: Maximum slack is inconsistent with a path that controls completion; no resource constraints are assumed. See PDF p. 95.

Primary note proof

Study-note section SE29

In a dependency network, ES=max predecessor EF and EF=ES+duration; start tasks have ES=0 unless another constraint applies. Project earliest finish is the maximum terminal EF. Backward, LF=min successor LS and LS=LF-duration, starting from the required finish. Total

float/slack=LS-ES=LF-EF. A critical path has the longest total duration and normally zero float in an unconstrained schedule.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 3-12

4.6. Scheduling, Organization and Team Structures
Project-task scheduling is an important project planning activity. It involves deciding which tasks would
be taken up when. In order to schedule the project activities, a software project manager needs to do
the following:
• Identify all the tasks needed to complete the project and Break down large tasks into small activities.
• Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
• Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Mock exam 2 · Question 32

Question 113 | Q01644

A zero-duration start A leads to parallel paths B(5)→C(6) and D(8)→E(3), both ending at a zero-duration finish. Which critical-path statements are true? Select all that apply. Multiple critical paths are allowed.

A. A→B→C is a critical path with total duration 11 days.

B. A→D→E is a critical path with total duration 11 days.

C. A→D→E is a critical path with total duration 14 days.

D. A→B→C is a critical path with total duration 14 days.

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer A, B: A→B→C is a critical path with total duration 11 days.; A→D→E is a critical path with total duration 11 days.

The B-C branch totals 5+6=11 days and the D-E branch totals 8+3=11 days. With zero-duration start and finish, both are equally longest complete paths, so both are critical and the project duration is eleven days. There is no rule requiring a unique critical path. The alternatives using fourteen days miscalculate the corresponding branch totals.

Why each choice is right or wrong

A - Correct: The B-C branch totals eleven and is critical.

B - Correct: The D-E branch also totals eleven and is critical.

C - Incorrect: Eight plus three is eleven, not fourteen.

D - Incorrect: Five plus six is eleven, not fourteen.

Primary note proof

Study-note section SE29

> In a dependency network, ES=max predecessor EF and EF=ES+duration; start tasks have ES=0 unless another constraint applies. Project earliest finish is the maximum terminal EF. Backward, LF=min successor LS and LS=LF-duration, starting from the required finish. Total float/slack=LS-ES=LF-EF. A critical path has the longest total duration and normally zero float in an unconstrained schedule.

Study-note section SE29

> If A takes 5 days and B=3,C=4 both follow A, project duration is 5+max(3,4)=9 days. B finishes at 8, but the project waits for C at 9. If paths B5+C6 and D8+E3 share a zero-duration start/end, both total 11; there are two critical paths, not one.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 3-12

> 4.6. Scheduling, Organization and Team Structures
> Project-task scheduling is an important project planning activity. It involves deciding which tasks would
> be taken up when. In order to schedule the project activities, a software project manager needs to do
> the following:
> • Identify all the tasks needed to complete the project and Break down large tasks into small activities.
> • Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
> • Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Mock exam 2 · Question 33

Question 114 | Q01645

How can market dynamics influence the evolution of requirements?

A. By introducing new competitors with innovative products

B. By stabilizing existing market conditions

C. By reducing the need for frequent updates

D. By minimizing consumer demands

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: By introducing new competitors with innovative products

Product competition is a business risk. A competitor introducing a product can alter customer needs and thus motivate evolution.

Why each choice is right or wrong

A - Correct: Direct concrete competitive stimulus consistent with the risk table on PDF p.97. See PDF p. 96, 41.

B - Incorrect: Stabilization does not supply the change stimulus described. See PDF p. 96, 41.

C - Incorrect: Reduced updates does not explain new competitive requirements. See PDF p. 96, 41.

D - Incorrect: Minimizing demands is not the described competitive pressure. See PDF p. 96, 41.

Primary note proof

Study-note section SE11

> Correct requirements reflect actual stakeholder needs. Complete requirements address the agreed scope, normal and exceptional behavior. Consistency excludes contradictions. Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable within constraints. Verifiability means objective evidence can check fulfillment. Replace “fast for most users” with a defined metric, workload and threshold.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 96 | lines 12-21

> 4.7. Risk Management
> • Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
> • A risk is a probability that some adverse circumstance will occur
> - Project risks affect schedule or resources;
> - Product risks affect the quality or performance of the software being developed;
> - Business risks affect the organisation developing or procuring the software.
> Risk Management Process

- Risk identification- Identify project, product and business risks;
- Risk analysis -Assess the likelihood and consequences of these risks;

Software Engineering module | PDF page 41 | lines 16-17

✓ Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.

Software Engineering module | PDF page 97 | lines 30-33

Technology change Business The underlying technology on which the system is built is superseded by new technology.
Product competition Business A competitive product is marketed before the system is completed.

Question sources

Mock exam 2 · Question 34

Question 115 | Q01646

The static nature of the system is represented by which model?

A. Context Model

B. Data Model

C. Structural Model

D. Behavioral Model

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Structural Model

The class/object model describes structure rather than temporal behavior; structural model is the offered general category.

Why each choice is right or wrong

A - Incorrect: Context concerns system boundaries, not all internal static structure. See PDF p. 51, 101.

B - Incorrect: Data is one aspect of structure but narrower than the overall static model. See PDF p. 51, 101.

C - Correct: Best matches the system's static classes/relationships. See PDF p. 51, 101.

D - Incorrect: Behavioral models describe interactions and changes over time. See PDF p. 51, 101.

Primary note proof

Study-note section SE17

A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links. UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process. Structural things describe conceptual or physical structural elements.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 51 | lines 14-24

2.3.2. Class Diagrams
Classes and objects
Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behavior of a set of objects. Objects are entities that encapsulate state and behavior. Each object has an identity: It can be referred individually and is distinguishable from other objects.
In UML, classes and objects are depicted by boxes including three compartments. The top compartment displays the name of the class or object. The center compartment displays its attributes, and the bottom compartment displays its operations. The attribute and operation compartment can be omitted for clarity. Object names are underlined to indicate that they are instances. By convention, class names start with an uppercase letter. Objects in object diagrams

Software Engineering module | PDF page 101 | lines 5-6

In UML, the functional model is represented with use case diagrams, the object model with class diagrams, and the dynamic model with statechart and sequence diagrams.

Question sources

Mock exam 2 · Question 35

Question 116 | Q01647

How does backward traceability help in requirements management?

A. By identifying redundant requirements

B. By mapping requirements to design documents

C. By ensuring that each requirement can be traced back to its source

D. By linking requirements to future project phases

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Trace each requirement back to its source and rationale.

SE11 directs requirements backward to their source or rationale and forward to design, code and tests. Backward traceability therefore establishes why a requirement exists and where it came from. It can assist reviews, but redundancy detection is not its definition. Design mappings and future phases primarily describe the forward direction.

Why each choice is right or wrong

A - Incorrect: Redundancy detection can benefit from traceability, but it is not the definition of backward tracing.

B - Incorrect: Mapping toward design documents is forward traceability from the requirement.

C - Correct: Backward tracing establishes the source or rationale from which each requirement originated.

D - Incorrect: Links toward later project artifacts describe forward rather than backward traceability.

Primary note proof

Study-note section SE11

> Trace each requirement backward to its source/rationale and forward to design, code and tests. Record dependencies among requirements. A traceability matrix links these artifacts and helps impact analysis; it is not just a cost/schedule ledger. Stakeholders must negotiate conflicting requirements rather than independently approving incompatible demands. Scope, understanding and volatility are distinct elicitation difficulties.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 70 | lines 3-16

> 3.2.5. Verifiability and Traceability
> Two more desirable properties of a system specification are that it be verifiable and traceable. The
> specification is verifiable if, once the system is built; a repeatable test can be designed to demonstrate that the system fulfills the requirement. For example, a mean time to failure of a hundred years for SatWatch would be difficult to achieve (assuming it is realistic in the first place).
> The following requirements are additional examples of non-verifiable requirements:
> • The product shall have a good user interface (good is not defined).
> • The product shall be error free (requires large amount of resources to establish).
> • The product shall respond to the user with 1 second for most cases (“most cases” is not defined).
> A system specification is traceable if each system function can be traced to its corresponding set of requirements. Traceability is not a constraint on the content of the specification, but rather, on its organization. Traceability facilitates the development of tests and the systematic validation of the design against the requirements.

Question sources

Mock exam 2 · Question 36

Question 117 | Q01648

Which artifact primarily serves project budgeting rather than specifying, planning or tracing tests?

A. Test strategy

B. Test Case

C. Project-wide cost estimate

D. Requirements Traceability Matrix [RTM]

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: Project-wide cost estimate

A test strategy describes how testing will be approached, a test case supplies actions or inputs with expected results, and an RTM links requirements to tests to support coverage and impact analysis. A project-wide cost estimate instead budgets the broader project. It may include testing costs and inform a test plan, but its primary purpose is project planning rather than specifying or tracing tests. The distractor uses that explicit distinction.

Why each choice is right or wrong

A - Incorrect: A test strategy defines the intended testing approach.

B - Incorrect: A test case specifies inputs or actions and expected results.

C - Correct: A project-wide cost estimate is a budgeting artifact, not primarily a testing specification or trace.

D - Incorrect: An RTM links requirements to tests and supports test coverage.

Primary note proof

Study-note section SE39

> Plan objectives, scope, resources, schedule, risks and entry/exit criteria. Analyze the test basis for testable features and defects; design cases/expected results; implement data, scripts and environment; execute and compare; report defects; evaluate completion.

Study-note section SE11

> Trace each requirement backward to its source/rationale and forward to design, code and tests. Record dependencies among requirements. A traceability matrix links these artifacts and helps impact analysis; it is not just a cost/schedule ledger.

Study-note section SE28

Planning is continuous: identify scope/deliverables, decompose work, estimate effort/duration/cost, assign resources, sequence dependencies and revise as evidence changes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Software Engineering module | PDF page 90 | lines 7-24

4.2. Project Planning
• Probably the most time-consuming project management activity.
• Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
• Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
The project plan sets out:
• The resources available to the project;
• The work breakdown;
• A schedule for the work.
Project Plan Structure
• Introduction.
• Project organisation.
• Risk analysis.
• Hardware and software resource requirements.
• Work breakdown.
• Project schedule.
• Monitoring and reporting mechanisms.

Question sources

Mock exam 2 · Question 37

Question 118 | Q01649

Degree to which design specifications are followed in manufacturing the product is called

A. Quality Control

B. None of the mentioned

C. Quality of conformance

D. Quality Assurance

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer C: Quality of conformance.

SE34 defines quality of conformance as following the design or specification. That is exactly the degree asked for in the stem. Quality control evaluates work products and quality assurance establishes and checks processes, but neither is the name of this conformance property. A conforming product may still follow an inadequate specification.

Why each choice is right or wrong

A - Incorrect: Quality control evaluates products rather than naming their degree of adherence to the design.

B - Incorrect: The exact property is offered as quality of conformance.

C - Correct: Conformance measures how faithfully the product follows its design or specification.

D - Incorrect: Quality assurance concerns processes and standards rather than the named degree of product conformance.

Primary note proof

Study-note section SE34

> Quality assurance establishes and checks processes/standards; quality control evaluates work products. An SQA plan identifies standards, responsibilities, evaluations, reviews, audits and assurance records. Quality of conformance means following the design/specification; a product can conform to a poor specification. Verification asks whether specified obligations are implemented correctly; validation asks whether the product meets intended stakeholder needs. Both can use reviews, analysis or execution, so “human-only versus computer-only” is false.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 99 | lines 14-19

> 4.8. Quality Assurance Monitoring Plans
> Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product

quality, thereby gaining insight and confidence that product quality is meeting its goals. Of course,
if the data provided through quality assurance identify problems, it is management's responsibility
to address the problems and apply the necessary resources to resolve quality issues.

Software Engineering module | PDF page 140 | lines 11-18

Reliability is a measure of success with which the observed behavior of a system conforms to the specification of its behavior. Software reliability is the probability that a software system will not cause the failure of the system for a specified time under specified conditions. Failure is any deviation of the observed behavior from the specified behavior. An error means the system is in a state such that further processing by the system will lead to a failure, which causes the system to
deviate from its intended behavior. A fault, also called defect or bug, is the mechanical or algorithmic cause of an error. The goal of testing is to maximize the number of discovered faults, which then allows developers to correct them and increase the reliability of the system.

Question sources

Mock exam 2 · Question 38

Question 119 | Q01650

Which of the below testing is related to Non-functional testing?

A. Performance Testing

B. Unit Testing

C. Black-box Testing

D. None of the choices are correct

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer A: Performance Testing

Performance testing explicitly checks nonfunctional requirements and additional design goals.

Why each choice is right or wrong

A - Correct: Directly tests nonfunctional qualities. See PDF p. 144.

B - Incorrect: A test level that can use several techniques and requirement types. See PDF p. 144.

C - Incorrect: A knowledge-based technique category, not specifically nonfunctional scope. See PDF p. 144.

D - Incorrect: The performance option is supported. See PDF p. 144.

Primary note proof

Study-note section SE36

Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair. Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Question sources

Mock exam 2 · Question 39

Question 120 | Q00034B

First design the very high level structure of the system. Then gradually work down to detailed decisions about low-level constructs. Which design approach follows the given strategy?

A. Bottom-up

B. System design

C. Top-down

D. Hybrid

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Top-down design.

The described direction starts with high-level structure and refines it into lower-level decisions. SE25 explicitly calls that top-down design. Bottom-up starts from existing components and assembles them; a hybrid combines directions. System design names a design activity or level, not the specific refinement direction requested by the question.

Why each choice is right or wrong

A - Incorrect: Bottom-up proceeds from components toward a larger structure, reversing the described direction.

B - Incorrect: System design names an architectural activity rather than the specific high-to-low refinement approach.

C - Correct: Top-down begins with high-level structure and progressively refines implementation details.

D - Incorrect: A hybrid combines both directions; the stem describes only top-down refinement.

Primary note proof

Study-note section SE25

> Use abstraction and decomposition to manage complexity; preserve consistency, traceability and clear responsibilities. Top-down design refines high-level structure; bottom-up design assembles existing components. A hybrid combines both. SOLID includes Single Responsibility, Open-Closed, Liskov substitution, Interface Segregation and Dependency Inversion. Depend on abstractions; add compatible extensions; avoid forcing unused operations. DRY limits duplicated knowledge, KISS avoids unnecessary complexity, and YAGNI avoids speculative work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 118 | lines 11-16

> 6.1.1. Subsystems and classes
> In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Software Engineering module | PDF page 18 | lines 13-20

> System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
> Software design involves identifying and describing the fundamental software system abstractions
> and their relationships. The goal of the design phase is to transform the requirements specified in
> the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from

the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Question sources

MOE Computer Science Exit Exam · 2016 · Question 34

Question 121 | SM0006

If you plan a software test to be done by clients to demonstrate that the system satisfies their requirements, which phase of software testing is appropriate?

A. Acceptance testing

B. Integration testing

C. System testing

D. Unit testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer A: Acceptance testing

Acceptance testing checks requirements against the project agreement and is performed by the client, matching both actor and purpose in the stem.

Why each choice is right or wrong

A - Correct: Direct client agreement/acceptance check. See PDF p. 145, 144.

B - Incorrect: Combines individually tested components. See PDF p. 145, 144.

C - Incorrect: Tests the complete system, but the specified client acceptance purpose identifies the narrower level. See PDF p. 145, 144.

D - Incorrect: Isolates building blocks, not client acceptance. See PDF p. 145, 144.

Primary note proof

Study-note section SE35

Acceptance testing supports the customer/user's acceptance decision against agreed needs and criteria. Alpha typically happens in a controlled developer setting with selected users; beta uses external users in their environments. None of these levels is universally the longest. Unit is a level, while white-box is a technique; a unit can be tested using black-box cases too.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 145 | lines 3-5

o Acceptance testing and installation testing check the requirements against the project agreement and should be done by the client, if necessary with support by the developers.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Question sources

MOE Computer Science Exit Exam · Question 2-2

National Computer Science Exit Exam · Question 46

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 1

Question 122 | SM0027

Assume you make modification to your software. Then you want to make sure that the modification
has not introduced new errors. Which software testing level is required?

A. Regression testing

B. Static testing

C. Beta testing

D. Dynamic testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer A: Regression testing.

Regression testing means checking for unintended effects after a software change. That matches the stated goal of finding newly introduced errors. Static and dynamic describe execution-related approaches rather than this change-focused purpose; beta describes an external-user test setting. Although the stem says level, regression is more accurately a testing purpose/type.

Why each choice is right or wrong

A - Correct: Regression testing checks whether a modification damages behavior that previously worked.

B - Incorrect: Static testing examines artifacts without execution and does not specifically define the change-impact purpose.

C - Incorrect: Beta testing involves external users in their environments rather than specifically checking every change's unintended effects.

D - Incorrect: Dynamic testing means executing software; it is broader than the regression purpose described.

Primary note proof

Study-note section SE36

> Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair. Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 147 | lines 14-18

> • Unit testing, which finds faults by isolating an individual component using test stubs and drivers and by exercising the component using a test case
> • Integration testing, which finds faults by integrating several components together
> • System testing, which focuses on the complete system, its functional and nonfunctional requirements, and its target environment

Software Engineering module | PDF page 142 | lines 5-10

> Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
> faults if change is controlled.

Software Engineering module | PDF page 147 | lines 3-8

• A test stub is a partial implementation of components on which the tested component depends. A test driver is a partial implementation of a component that depends on the tested component. Test stubs and drivers enable components to be isolated from the rest of the system for testing.
• A correction is a change to a component. The purpose of a correction is to repair a fault. Note that a correction can introduce new faults.

Question sources

MOE Computer Science Exit Exam · Question 8-2

National Computer Science Exit Exam · Question 54

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 2

Question 123 | SM0066

After deep analysis if your software requires major change to the architecture of the system, which
strategy of software change should be applied?

A. Preventative maintenance

B. Architectural transformation

C. Software reengineering

D. Software maintenance

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer B: Architectural transformation.

SE40 names major restructuring reengineering and identifies architectural transformation as the more specific term when the architecture itself changes. The stem expressly requires major architecture change, making B the best match. Maintenance and reengineering are broader categories, while preventive maintenance describes a motive and need not involve architectural restructuring.

Why each choice is right or wrong

A - Incorrect: Preventive maintenance reduces future risk but does not specifically denote a major architectural change.

B - Correct: Architectural transformation directly names the major architecture change required by the scenario.

C - Incorrect: Reengineering is a broader restructuring category; architectural transformation is more specific here.

D - Incorrect: Maintenance is the broad activity family rather than the particular architecture-change strategy.

Primary note proof

Study-note section SE40

A typical evolution sequence is change request→impact analysis→release planning→implementation/testing→release. Among a list lacking the request step, impact analysis may be the earliest listed technical activity. Major restructuring is reengineering; if the architecture itself changes, architectural transformation is the more specific description. Markets, technology, stakeholders and funding can all change scope or priorities.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 121 | lines 6-15

6.1.4. Software architecture
As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
In this section, we describe a few sample architectures that can be used for different systems. This
is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
representative examples and refer the reader to the literature for more details.

Software Engineering module | PDF page 19 | lines 24-29

5. Operation and maintenance: Normally (although not necessarily), this is the longest life cycle phase. Maintenance of a typical software product requires much more than the effort necessary to
develop the product itself. The system is installed and put into practical use. Maintenance involves
correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system ‘s services as new requirements are discovered.

Question sources

MOE Computer Science Exit Exam · Question 19-2

Computer Science Exit Exam practice collection · Question 3

Question 124 | SM0078

Which one of the following does NOT show a non-functional requirement of software systems?

A. Response time

B. Memory requirement

C. Reliability

D. Displaying information

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: Displaying information

Response time, memory constraints and reliability are nonfunctional qualities. Displaying information specifies a service rather than a quality bound.

Why each choice is right or wrong

A - Incorrect: Quantitative speed constraint. See PDF p. 65, 66, 127.

B - Incorrect: Space/resource constraint. See PDF p. 65, 66, 127.

C - Incorrect: Dependability quality. See PDF p. 65, 66, 127.

D - Correct: Observable function of presenting information. See PDF p. 65, 66, 127.

Primary note proof

Study-note section SE09

> A functional requirement states an observable service or behavior, including reactions to inputs, errors and forbidden actions. “When the sensor detects intrusion, send an alert” is functional. “Only a manager may approve refunds” also specifies permitted/prohibited behavior. A negative statement is not automatically nonfunctional.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 65 | lines 19-23

> 3.2.1. Functional Requirements
> Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. For example, the following is an example of functional requirements for SatWatch, a watch that resets itself without user intervention:

Software Engineering module | PDF page 66 | lines 14-18

> 3.2.2. Non-Functional and Pseudo-Requirements
> Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Nonfunctional requirements include

quantitative constraints, such as response time (i.e., how fast the system reacts to user commands)
or accuracy (i.e., how precise are the system's numerical answers). The following are the

Software Engineering module | PDF page 127 | lines 10-28

6.2.1. Identifying design goals
The definition of design goals is the first step of system design. It identifies the qualities that our system should focus on. Many design goals can be inferred from the nonfunctional requirements or from the application domain. Others will have to be elicited from the client. It is, however, necessary to state them explicitly such that every important design decision can be made consistently following the same set of criteria.
For example, in the light of the nonfunctional requirements for MyTrip, we identify reliability and
fault tolerance to connectivity loss as design goals. We then identify security as a design goal, as numerous drivers will have access to the same trip planning server. We add modifiability as a design goal, as we want to provide the ability for drivers to select a trip planning service of their choice. The following box summarizes the design goals we identified.
In general, we can select design goals from a long list of highly desirable qualities. Tables 5-1-5-5 list a number of possible design criteria. These criteria are organized into five groups: performance, dependability, cost, maintenance, and end user criteria. Performance, dependability,
and end user criteria are usually specified in the requirements or inferred from the application domain. Cost and maintenance criteria are dictated by the customer and the supplier.
Performance criteria (Table 5-1) include the speed and space requirements imposed on the system. Should the system be responsive or should it accomplish a maximum number of tasks? Is memory space available for speed optimizations or should memory be used sparingly?

Question sources

MOE Computer Science Exit Exam · Question 22-3

National Computer Science Exit Exam · Question 45

National Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 4

Question 125 | SM0079

Which one of the following is false about project planning stage?

A. Risk analysis is done during project planning

B. Preparing time schedule is done during project planning

C. Cost estimation is done during project planning

D. Project planning is a one-time task in software development life cycle

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer D: Project planning is a one-time task in software development life cycle

Planning is explicitly continuous from initial concept through delivery and revised as information changes. Therefore the one-time statement is false.

Why each choice is right or wrong

A - Incorrect: Risk analysis is in the plan structure. See PDF p. 90.

B - Incorrect: Schedule is in the project plan. See PDF p. 90.

C - Incorrect: Plans concern schedule and budget, so cost estimation belongs in planning. See PDF p. 90.

D - Correct: Direct contradiction of continuous revision. See PDF p. 90.

Primary note proof

Study-note section SE28

> Planning is continuous: identify scope/deliverables, decompose work, estimate effort/duration/cost, assign resources, sequence dependencies and revise as evidence changes. A WBS is hierarchical and covers project scope without adding unrelated work. Tools help coordinate ownership, dependencies and progress; they do not replace judgment or automatically fix bugs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 90 | lines 7-24

> 4.2. Project Planning
> • Probably the most time-consuming project management activity.
> • Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
> • Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
> The project plan sets out:
> • The resources available to the project;
> • The work breakdown;
> • A schedule for the work.
> Project Plan Structure
> • Introduction.
> • Project organisation.
> • Risk analysis.
> • Hardware and software resource requirements.
> • Work breakdown.

• Project schedule.
• Monitoring and reporting mechanisms.

Question sources

MOE Computer Science Exit Exam · Question 22-4

National Computer Science Exit Exam · Question 14

National Computer Science Exit Exam

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 5

Question 126 | SM0090

Which phase of software development process refers to modification of software to reflect changing customer and market requirements?

A. Validation

B. Evolution

C. Development

D. Specification

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Evolution

Changing existing software to meet new requirements is software evolution.

Why each choice is right or wrong

A - Incorrect: Checks conformance and customer expectations. See PDF p. 41.

B - Correct: Exact ongoing change activity. See PDF p. 41.

C - Incorrect: Produces a software implementation. See PDF p. 41.

D - Incorrect: Defines required services and constraints. See PDF p. 41.

Primary note proof

Study-note section SE01

Software engineering applies systematic methods to development, operation and maintenance. A software product includes executable code and relevant requirements, designs, test assets, configuration and user documentation. A process defines activities, ordering, roles, products and entry/exit conditions. Its four broad activities are specification, design/implementation, validation and evolution.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 41 | lines 16-17

> ✓ Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.

Question sources

MOE Computer Science Exit Exam · Question 25-3

MOE Computer Science Exit Exam

Computer Science Exit Exam practice collection · Question 6

Computer Science Exit Exam practice collection · Question 11

Question 127 | SM0203

When is it appropriate to use the Spiral model?

A. Where business modeling is clearly understood and in use

B. When it is too risky to develop the overall system as one block

C. When requirements are understood early but can change during the project's life cycle

D. When requirements are unstable and constantly changing

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: When major risks make one-block development unsafe.

SE06 uses explicit risk assessment and reduction to distinguish spiral development from other iterative approaches. B directly states that risk makes a single overall commitment inappropriate. Changing or unstable requirements can occur in several processes and do not alone identify spiral; clear business modeling likewise does not supply the distinctive risk-based reason.

Why each choice is right or wrong

A - Incorrect: Understanding business modeling alone does not identify the need for risk-driven spiral cycles.

B - Correct: Important risks justify staged investigation and commitment rather than building the entire system in one block.

C - Incorrect: Possible requirement changes fit multiple processes and omit spiral's distinguishing explicit risk emphasis.

D - Incorrect: Unstable requirements alone do not uniquely select spiral over other adaptive approaches.

Primary note proof

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Study-note section SE06

> Use risk emphasis to distinguish spiral from a generic changing-requirements scenario.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 96 | lines 12-21

> 4.7. Risk Management
> • Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
> • A risk is a probability that some adverse circumstance will occur
> - Project risks affect schedule or resources;
> - Product risks affect the quality or performance of the software being developed;
> - Business risks affect the organisation developing or procuring the software.
> Risk Management Process
> - Risk identification- Identify project, product and business risks;
> - Risk analysis -Assess the likelihood and consequences of these risks;

Software Engineering module | PDF page 20 | lines 18-22

> the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
> In principle, the waterfall model should only be used when the requirements are well understood
> and unlikely to change radically during system development. However, the waterfall model

Software Engineering module | PDF page 22 | lines 8-12

> This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
> also provide a platform for evaluation by the user.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 1-3

Computer Science Exit Exam practice collection · Question 14

Question 128 | SM0235

A module is said to have logical cohesion if:

A. All the functions of the module are executed within the same time span

B. All elements of the module depend on their interface complexity

C. It performs a set of tasks that relate to each other very loosely

D. All elements of the module perform similar operations

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: The module groups similar-category operations.

SE22 defines logical cohesion as grouping operations in a similar category, often selected through a flag. D matches that shared logical category. A instead describes temporal cohesion, where operations occur at the same time. Interface complexity concerns dependencies, and a loose collection of unrelated tasks does not state the defining category-based relationship.

Why each choice is right or wrong

A - Incorrect: Grouping operations because they occur at the same time is temporal cohesion.

B - Incorrect: Interface complexity concerns dependencies and does not define logical cohesion inside a module.

C - Incorrect: Loose task relatedness alone does not establish the shared operation category defining logical cohesion.

D - Correct: Grouping similar-category operations is the defining feature of logical cohesion.

Primary note proof

Study-note section SE22

> Logical cohesion groups similar-category operations, often selected by a flag; temporal cohesion groups operations done at the same time; functional cohesion focuses on one task. Splitting an unrelated three-purpose component improves modularity. If behavior is preserved, the restructuring is refactoring; the words are not mutually exclusive answers. Smaller modules can aid understanding, independent work, reuse and maintenance, but too many interfaces also create complexity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 120 | lines 24-27

> Coherence is the strength of dependencies within a subsystem. If a subsystem contains many objects that are related to each other and perform similar tasks, its coherence is high. If a subsystem

contains a number of unrelated objects, its coherence is low. A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 11-3

Computer Science Exit Exam practice collection · Question 17

Question 129 | SM0244

Which software engineering principles are used to help reduce programming complexity?

A. Structured and unstructured

B. Object Oriented Design

C. Abstraction and Decomposition

D. Functional and Non-functional

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Abstraction and Decomposition

Abstraction lets developers reason about objects at a higher level while decomposition divides a complex system into simpler subsystems; both directly control complexity.

Why each choice is right or wrong

A - Incorrect: Structured/unstructured names do not identify the paired techniques. See PDF p. 31, 118.

B - Incorrect: OO design employs these mechanisms but is the broader methodology. See PDF p. 31, 118.

C - Correct: Combines the two explicit complexity-management mechanisms. See PDF p. 31, 118.

D - Incorrect: These classify requirements, not techniques for decomposing a program. See PDF p. 31, 118.

Primary note proof

Study-note section SE22

Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module's internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 31 | lines 19-31

1.3.1. Why an object oriented
• Object oriented systems are
o Easier to adapt to changes
o More robust
o Easier to maintain
o Promote greater design and code reuse
o Creates modules of functionality
• REASONS FOR WORKING OF OBJECT ORIENTED SYSTEMS:
Higher level of abstraction
The object-oriented approach supports abstraction at the object level. Since objects encapsulate both data (attributes) and functions (methods), they work at a higher level of abstraction. The development can proceed at the object level and ignore the rest of the system for as long as necessary. This makes designing, coding, testing, and maintaining the system much simpler.

Software Engineering module | PDF page 118 | lines 11-16

6.1.1. Subsystems and classes
In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Question sources

Computer Science Exit Exam · Sene 2016 · Question 14-3

Computer Science Exit Exam practice collection · Question 15

Question 130 | SM0271

Which listed model organizes repeated cycles around risk assessment and can select prototyping, incremental or sequential activities as appropriate within those cycles?

A. Spiral model

B. Prototyping model

C. Classical Waterfall model

D. Iterative Waterfall model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Spiral model

Spiral development repeatedly identifies objectives, assesses and reduces risks, develops an appropriate result and plans further commitment. Its cycles can incorporate prototyping, incremental work or sequential activities according to the risks. The other listed models name particular exploratory or staged organizations rather than this risk-driven choice framework. This explains the conventional general-model reading without claiming that every process is formally a specialization of spiral.

Why each choice is right or wrong

A - Correct: Spiral selects cycle work through risk assessment.

B - Incorrect: Prototyping is one learning activity that a spiral cycle may use.

C - Incorrect: Classical waterfall is a planned sequential organization.

D - Incorrect: Iterative waterfall adds feedback without defining the risk-driven cycle.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 20 | lines 18-22

> the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
> In principle, the waterfall model should only be used when the requirements are well understood
> and unlikely to change radically during system development. However, the waterfall model

Software Engineering module | PDF page 22 | lines 8-12

> This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user

and
also provide a platform for evaluation by the user.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 23-2

Computer Science Exit Exam practice collection · Question 16

Question 131 | SM0293

Which system requirement specification includes human-computer interface issues from the system?

A. Functional requirements of the system

B. Non-functional requirements of the system

C. Goals of implementation

D. System Testing

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Non-functional requirements of the system

User-interface look/feel and user expertise are explicitly nonfunctional concerns, under user interface and human factors. Particular interaction services can still be functional; the question is read as interface quality issues.

Why each choice is right or wrong

A - Incorrect: Specific interface functions can be functional, but this is not the broad human-factor quality category. See PDF p. 78, 66.

B - Correct: Explicitly includes user interface and human factors. See PDF p. 78, 66.

C - Incorrect: Implementation goals are not the stated requirement category. See PDF p. 78, 66.

D - Incorrect: Testing is an activity, not a requirement category. See PDF p. 78, 66.

Primary note proof

Study-note section SE10

Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning. Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 78 | lines 12-30

3.3.7. Identifying Non-Functional Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related
to the functional behavior of the system. Nonfunctional requirements span a number of issues, from user interface look and feel to response time requirements to security issues. Nonfunctional
requirements are defined at the same time as functional requirements are, because they have as much impact on the development and cost of the system.
For example, consider a mosaic display that an air traffic controller uses to track planes. A mosaic
display system complies data from a series of radar and databases (hence the term “mosaic”) into
a summary display indicating all aircraft in a certain area, including their identification, speed, and
altitude. The number of aircraft such a system can display constrains the performance of the air traffic controller and the cost of the system. If the system can only handle a few aircraft simultaneously, the system cannot be used at busy airports. On the other hand, a system able to handle a large number of aircraft is more costly and more complex to build.
Nonfunctional requirements can be elicited by investigating the following issues.
• User interface and human factors. What kind of interface should the system provide? What is the level of expertise of the users?
• Documentation. What level of document is required? Should only user documentation be provided? Should there be technical documentation for maintainers? Should the development process be documented?

Software Engineering module | PDF page 66 | lines 14-18

3.2.2. Non-Functional and Pseudo-Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Nonfunctional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands)
or accuracy (i.e., how precise are the system’s numerical answers). The following are the

Question sources

Computer Science Exit Exam · Sene 2016 · Question 30-3

Computer Science Exit Exam practice collection · Question 18

Question 132 | SM0294

In a classical waterfall model, which phase precedes the design phase?

A. Requirements analysis and specification

B. Maintenance

C. Feasibility study

D. Coding and unit testing

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Requirements analysis and specification

The sequential waterfall establishes the requirements specification before deriving the software design from the SRS.

Why each choice is right or wrong

A - Correct: The immediate input and preceding phase. See PDF p. 15, 18, 26.

B - Incorrect: Occurs after delivery and operation. See PDF p. 15, 18, 26.

C - Incorrect: A prior viability activity, not the immediate specification input to design. See PDF p. 15, 18, 26.

D - Incorrect: Implements a design after it has been produced. See PDF p. 15, 18, 26.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 15 | lines 16-20

> non-iterative (or sequential) process in which progress is seen as flowing steadily downwards (like
> a waterfall) through the development phases. Thus the waterfall model maintains that one should
> move to a phase only when its preceding phase is reviewed and verified. That is, each phase must
> be completed before the next phase can begin. In waterfall model phases do not overlap. But

there
are various modified waterfall models, and hence, may include slight or major modifications.

Software Engineering module | PDF page 18 | lines 13-20

System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
Software design involves identifying and describing the fundamental software system abstractions
and their relationships. The goal of the design phase is to transform the requirements specified in
the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Software Engineering module | PDF page 26 | lines 10-17

3. Requirement's specification: Requirements specification is the activity of translating the information gathered during the analysis activity in to a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are
abstract statements of the system requirements for the customer and end-user of the system; system
requirements are a more detailed description of the functionality to be provided.
4. Requirement's validation: This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered.
It must then be modified to correct these problems.

Question sources

Computer Science Exit Exam · Sene 2016 · Question 31-1

Computer Science Exit Exam practice collection · Question 19

Question 133 | SM0300

When to use the Spiral Model?

A. Where business modeling is clearly understood and in use.

B. When it is too risky to develop the overall system as one block.

C. When requirements are understood in the early stages but can change during the project's life cycle.

D. When requirements are unstable and constantly changing.

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: When major risks make whole-system development in one block unsafe.

A spiral cycle identifies objectives, assesses and reduces important risks, develops an appropriate result and plans further commitment. B directly supplies the risk-based reason for choosing that process. Clear business understanding, potentially changing requirements and continuously unstable requirements do not alone distinguish spiral from other staged or adaptive models. The explicit risk is the decisive feature.

Why each choice is right or wrong

A - Incorrect: Clear business modeling does not by itself create a reason for risk-driven spiral development.

B - Correct: Risk makes staged assessment and commitment preferable to building the whole system in one block.

C - Incorrect: Requirements that may change fit several processes and do not specifically identify spiral.

D - Incorrect: Unstable requirements alone do not distinguish spiral without its explicit risk-analysis need.

Primary note proof

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 96 | lines 12-21

> 4.7. Risk Management
> • Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
> • A risk is a probability that some adverse circumstance will occur
> - Project risks affect schedule or resources;
> - Product risks affect the quality or performance of the software being developed;
> - Business risks affect the organisation developing or procuring the software.
> Risk Management Process
> - Risk identification- Identify project, product and business risks;
> - Risk analysis -Assess the likelihood and consequences of these risks;

Software Engineering module | PDF page 20 | lines 18-22

the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
In principle, the waterfall model should only be used when the requirements are well understood
and unlikely to change radically during system development. However, the waterfall model

Question sources

Computer Science Exit Exam · Sene 2016 · Question 32-3

Question 134 | SS096

The process that generates activities such as project schedule, cost estimations and work breakdown structures is described as:

A. Project Executing

B. Project Initiating

C. Project Closing

D. Project Planning

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer D: Project Planning

The project plan explicitly sets out resources, work breakdown and schedule and concerns budget; these are outputs of project planning.

Why each choice is right or wrong

A - Incorrect: Execution carries out planned tasks. See PDF p. 90, 95.

B - Incorrect: Initiation frames a project but does not best identify this full planning output set. See PDF p. 90, 95.

C - Incorrect: Closing concerns completion/acceptance, not scheduling future work. See PDF p. 90, 95.

D - Correct: Directly corresponds to the listed plan contents. See PDF p. 90, 95.

Primary note proof

Study-note section SE28

Planning is continuous: identify scope/deliverables, decompose work, estimate effort/duration/cost, assign resources, sequence dependencies and revise as evidence changes. A WBS is hierarchical and covers project scope without adding unrelated work. Tools help coordinate ownership, dependencies and progress; they do not replace judgment or automatically fix bugs.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 90 | lines 7-24

4.2. Project Planning
• Probably the most time-consuming project management activity.
• Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
• Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
The project plan sets out:
• The resources available to the project;
• The work breakdown;
• A schedule for the work.
Project Plan Structure
• Introduction.
• Project organisation.
• Risk analysis.
• Hardware and software resource requirements.
• Work breakdown.
• Project schedule.
• Monitoring and reporting mechanisms.

Software Engineering module | PDF page 95 | lines 3-12

4.6. Scheduling, Organization and Team Structures
Project-task scheduling is an important project planning activity. It involves deciding which tasks would
be taken up when. In order to schedule the project activities, a software project manager needs to do
the following:
• Identify all the tasks needed to complete the project and Break down large tasks into small activities.
• Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
• Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Computer Science model exam collection · Question 3

Question 135 | SS103

Which slogan is against the agile method philosophy?

A. Working software over comprehensive documentation

B. Following a plan over responding to changes

C. Individuals and interactions over processes and tools

D. Customer collaboration over contract negotiation

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Following a plan over responding to changes

Agile values working software, people interacting, customer collaboration and responsiveness to change over the respective items on the other side of each comparison. It still recognizes value in documentation, tools, contracts and plans. The reversed preference for following a plan over responding to changes conflicts with this ordering. The other three alternatives retain the Agile ordering and therefore are not slogans against its philosophy.

Why each choice is right or wrong

A - Incorrect: This preserves the Agile preference for working software while still valuing documentation.

B - Correct: This reverses Agile’s preference for responding to change over following a plan.

C - Incorrect: This preserves the preference for individuals and interactions.

D - Incorrect: This preserves the preference for customer collaboration.

Primary note proof

Study-note section SE05

> Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to

follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Question sources

Computer Science model exam collection · Question 10

Question 136 | SS105

What would be the most likely consequence if every software stakeholder were rushing to be approved for their requirements without discussing with
other stakeholders?

A. Results in excess requirement

B. Attains the project schedule

C. Requirements become unambiguous

D. Leads to conflicting requirement

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: Leads to conflicting requirement

JAD brings stakeholders together to negotiate a mutually acceptable specification; without discussion, incompatible local demands can remain unresolved.

Why each choice is right or wrong

A - Incorrect: Excess scope is possible but not the most direct consequence of unnegotiated differences. See PDF p. 81, 68.

B - Incorrect: Rushing approvals does not prove the schedule is met. See PDF p. 81, 68.

C - Incorrect: Clarity is not obtained automatically by separate approvals. See PDF p. 81, 68.

D - Correct: Competing viewpoints can yield contradictions, precisely what consensus and consistency checks address. See PDF p. 81, 68.

Primary note proof

Study-note section SE11

Correct requirements reflect actual stakeholder needs. Complete requirements address the agreed scope, normal and exceptional behavior. Consistency excludes contradictions. Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable within constraints. Verifiability means objective evidence can check fulfillment. Replace “fast for most users” with a defined metric, workload and threshold.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 81 | lines 11-21

3.4.2. Negotiating specifications with clients: Joint Application Design
Joint Application Design (JAD) is effectiveness lies in that the requirements elicitation work is done in one single workshop session in which all stakeholders participate. Users, clients, developers, and a trained session leader sit together in one room to present their viewpoint, listen
to other viewpoints, negotiate, and agree on a mutually acceptable solution. The outcome of the workshop, the final JAD document, is a complete system specification document that includes definitions of data elements, work flows, and interface screens. Because the final document is jointly developed by the stakeholders (that is, the participants who not only have an interest in the
success of the project, but also can make substantial decisions) the final JAD document represents
an agreement between users, clients, and developers, and thus minimizes requirements changes later in the development process.

Software Engineering module | PDF page 68 | lines 18-26

system specification. Requirement validation involves checking if the specification is correct, complete, consistent, unambiguous, and realistic. A specification is correct if it represents the client's view of the system (i.e., everything in the requirements model accurately represents an aspect of the system). It is complete if all possible scenarios through the system are described, including exceptional behavior (i.e., all aspects of the system are represented in the requirements
model). The system specification is consistent if it does not contradict itself. The system specification is unambiguous if exactly one system is defined (i.e., it is not possible to interpret the specification two or more different ways). Finally, it is realistic if the system can be implemented within constraints. These properties are illustrated with UML instance diagrams in

Question sources

Computer Science model exam collection · Question 12

Question 137 | SS113

If the estimate for total discounted benefits for a project is Birr 120,000 and total discounted cost is Birr 100,000, what is the estimated return on
investment (ROI) percentage?

A. 30%

B. 12%

C. 10%

D. 20%

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer D: 20%.

The benefits and costs are already discounted to a common date, so use SE31 directly: ROI=(120000−100000)/100000×100%. Net benefit is Birr 20000, and dividing by Birr 100000 gives 0.20, or 20%. Dividing total benefits by costs would report a benefit-cost ratio, not the net return percentage requested.

Why each choice is right or wrong

A - Incorrect: Thirty percent does not follow from the given net benefit and cost.

B - Incorrect: Twelve percent is not 20000 divided by 100000 as a percentage.

C - Incorrect: Ten percent understates the calculated net return of twenty percent.

D - Correct: Net benefit is Birr 20000, which is 20% of the Birr 100000 cost.

Primary note proof

Study-note section SE31

> With benefits and costs already discounted to a common date, ROI=(benefits-costs)/costs*100%. Birr120000 benefits and Birr100000 costs give 20%, not 120%. Identify project risks to resources/schedule, product risks to quality, and business risks to the organization. Analyze likelihood and impact, plan responses and monitor.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 26 | lines 3-5

> effective from a business point of view and if it can be developed within existing budgetary constraints. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether or not to go ahead with a more detailed analysis.

Software Engineering module | PDF page 90 | lines 7-24

> 4.2. Project Planning
> • Probably the most time-consuming project management activity.
> • Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
> • Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
> The project plan sets out:
> • The resources available to the project;
> • The work breakdown;
> • A schedule for the work.
> Project Plan Structure
> • Introduction.

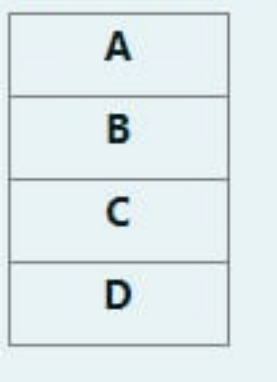
- Project organisation.
- Risk analysis.
- Hardware and software resource requirements.
- Work breakdown.
- Project schedule.
- Monitoring and reporting mechanisms.

Question sources

Computer Science model exam collection · Question 20

Question 138 | SS114

In a closed layered architecture with A,B,C,D from top to bottom, a pair (X,Y) means X directly calls Y. Calls may go only to the immediately lower layer. Which pair obeys this policy?



A. (A,C)

B. (D,A)

C. (A,B)

D. (C,B)

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: (A,B)

Closed layering permits a layer to call only the immediately lower layer. A→B follows that adjacency and direction, so C is valid. A→C skips B and violates the closed policy even though it would be possible in a suitably open architecture. D→A and C→B point upward, which the stipulated dependency policy also forbids. The ordered pair is interpreted as caller then callee.

Why each choice is right or wrong

A - Incorrect: A→C skips B.

B - Incorrect: D→A points upward.

C - Correct: A→B calls the immediately lower layer.

D - Incorrect: C→B points upward.

Primary note proof

Study-note section SE23

A layer is a logical abstraction level; a tier is a deployment separation. A closed layered architecture permits calls only to the immediately lower layer; an open one permits approved skipping. For top-to-bottom A,B,C,D, A→B obeys closed layering, A→C skips, and lower-to-upper calls violate the simple dependency rule unless explicitly mediated by a defined mechanism.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 121 | lines 6-15

6.1.4. Software architecture
As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
In this section, we describe a few sample architectures that can be used for different systems. This
is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
representative examples and refer the reader to the literature for more details.

Question sources

Computer Science model exam collection · Question 21

Question 139 | SS124

Which listed roles should document a fault if they discover it during development? Select all that apply.

A. Tester

B. Requirement engineer

C. Developer

D. Scrum master

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer A, B, C, D: Tester; Requirement engineer; Developer; Scrum master

Anyone discovering a fault should record it with enough evidence to reproduce and assess it. Testers often coordinate reporting, but requirements engineers, developers and Scrum masters should also document faults they discover. The report should identify the relevant environment, steps and expected versus actual behavior. Therefore all four roles meet the stated discoverer-based responsibility rather than only the dedicated tester.

Why each choice is right or wrong

A - Correct: A tester discovering a fault should record it.

B - Correct: A requirements engineer discovering a fault should record it.

C - Correct: A developer discovering a fault should record it.

D - Correct: A Scrum master discovering a fault should record it.

Primary note proof

Study-note section SE39

> Plan objectives, scope, resources, schedule, risks and entry/exit criteria. Analyze the test basis for testable features and defects; design cases/expected results; implement data, scripts and environment; execute and compare; report defects; evaluate completion. Analysis asks what to test and assesses testability; design asks how. Monitoring/control run across these activities.

Study-note section SE39

> Anyone finding a fault should record it; testers often coordinate reporting, but it is not their exclusive permission.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 24-33

> A review is the manual inspection of parts or all aspects of the system without actually executing the system. There are two types of reviews: walkthrough and inspection. In a code walkthrough, the developer informally presents the API, the code, and associated documentation of the component to the review team. The review team makes comments on the mapping of the analysis
> and object design to the code using use cases and scenarios from the analysis phase. An inspection
> is similar to a walkthrough, but the presentation of the unit is formal. In fact, in a code inspection,
> the developer is not allowed to present the artifacts (models, code, and documentation). This is done by the review team, which is responsible for checking the interface and code of the component against the requirements. It also checks the algorithms for efficiency with respect to the nonfunctional requirements. Finally, it checks comments about the code and compares them

Software Engineering module | PDF page 140 | lines 27-32

7.1 OVERVIEW OF SOFTWARE QUALITY ASSURANCE
We define testing as the systematic attempt to find errors in a planned way in the implemented software. Contrast this definition with another commonly used definition that says that “testing is
the process of demonstrating that errors are not present.” The distinction between these two definitions is important. Our definition does not mean we simply demonstrate that the program does what it is intended to do. The explicit goal of testing is to demonstrate the presence of faults.

Question sources

Computer Science model exam collection · Question 31

Question 140 | SS127

Select the one that is not correct related to generalization in system modeling.

A. Ingeneralization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes

B. Facilitate easy modification of data

C. The higher-level classes are more specific than the lower-level classes by adding specific attributes and operations

D. Common information will be maintained in one place

Hint

Match the information to be represented with each model’s purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: The higher-level classes are more specific than the lower-level classes by adding specific attributes and operations

Generalization puts common features in more general superclasses; subclasses add or override details and are more specialized. C reverses the direction.

Why each choice is right or wrong

A - Incorrect: Inherited attributes and methods are available on subclasses. See PDF p. 104.

B - Incorrect: Common definitions can reduce duplicated modification; the wording is broad. See PDF p. 104.

C - Correct: Higher-level classes are more general, not more specific. See PDF p. 104.

D - Incorrect: Consolidating common features in a superclass removes redundancy. See PDF p. 104.

Primary note proof

Study-note section SE18

Association connects related instances. A dashed dependency arrow means a client uses a supplier. Generalization uses a solid line and hollow triangle pointing to the superclass/general

element. Shared aggregation uses a hollow diamond at the whole end and weak whole-part meaning; it does not by itself define a detailed lifecycle policy.

Study-note section SE-S02

Generalization places common concepts in a more general superclass and specialized concepts in subclasses. Available inherited attributes and operations are shared by subclasses, which may extend or override behavior. Defining common features once can reduce duplicated maintenance. The top of the hierarchy is more general, not more specialized.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 104 | lines 6-17

5.1.4. Generalization
Generalization enables us to organize concepts in to hierarchies. There may be any number of intermediate levels in between, covering more-or-less generalized concepts (e.g., Low Priority Incident, Emergency, Disaster). Such hierarchies allow us to refer to many concepts precisely. When
we use the term Incident, we mean all instances of all types of Incidents. When we use the term Emergency, we only refer to an Incident that requires an immediate response. This view of generalization stems from modeling.
In an object-oriented programming language, inheritance is a reusability technique. If a class Child
inherits from a class Parent, all the attributes and methods available on Parent are automatically available on Child. The class Child may add additional methods or override inherited methods, thus refining the class Parent. As in the case of generalization, the class at the top of the hierarchy
tends to be the most general one, whereas the leaves tend to be the most specialized.

Question sources

Computer Science model exam collection · Question 34

Question 141 | SS135

Which statement accurately describes the main evidential goal of software testing within quality assurance?

A. To achieve project timelines and deadlines.

B. To monitor the entire SDLC.

C. To generate software requirement and design documentation.

D. To assess conformance to specified requirements, reveal defects and increase confidence through evidence.

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer D: To assess conformance to specified requirements, reveal defects and increase confidence through evidence.

Testing evaluates behavior against required results, revealing failures and increasing confidence in conformance. It does not generally guarantee correctness over every possible input or environment. Meeting schedules and monitoring the whole lifecycle are broader management or assurance activities, while requirements and design documentation usually form part of the test basis. Option D states the supported evidential purpose without promising certainty.

Why each choice is right or wrong

A - Incorrect: Timelines are a project-management goal rather than the test evidence goal.

B - Incorrect: Whole-lifecycle monitoring is broader assurance/control work.

C - Incorrect: Requirements and design documents are typically inputs to testing.

D - Correct: Tests assess conformance, reveal defects and increase confidence.

Primary note proof

Study-note section SE34

> Quality assurance establishes and checks processes/standards; quality control evaluates work products. An SQA plan identifies standards, responsibilities, evaluations, reviews, audits and assurance records. Quality of conformance means following the design/specification; a product can conform to a poor specification. Verification asks whether specified obligations are implemented correctly; validation asks whether the product meets intended stakeholder needs. Both can use reviews, analysis or execution, so “human-only versus computer-only” is false.

Study-note section SE34

> Tests increase confidence and reveal defects but cannot generally guarantee correctness over every possible input.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 140 | lines 27-32

> 7.1 OVERVIEW OF SOFTWARE QUALITY ASSURANCE
> We define testing as the systematic attempt to find errors in a planned way in the implemented software. Contrast this definition with another commonly used definition that says that “testing is
> the process of demonstrating that errors are not present.” The distinction between these two definitions is important. Our definition does not mean we simply demonstrate that the program does what it is intended to do. The explicit goal of testing is to demonstrate the presence of faults.

Software Engineering module | PDF page 140 | lines 11-18

Reliability is a measure of success with which the observed behavior of a system conforms to the specification of its behavior. Software reliability is the probability that a software system will not cause the failure of the system for a specified time under specified conditions. Failure is any deviation of the observed behavior from the specified behavior. An error means the system is in a state such that further processing by the system will lead to a failure, which causes the system to
deviate from its intended behavior. A fault, also called defect or bug, is the mechanical or algorithmic cause of an error. The goal of testing is to maximize the number of discovered faults, which then allows developers to correct them and increase the reliability of the system.

Question sources

Computer Science model exam collection · Question 42

Question 142 | SS137

What component hides the distinction or boundaries between various microservices from end-client application?

A. Alayered system

B. API gateway

C. API logging

D. API proxy

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: API gateway.

SE24 defines an API gateway as one entry point that routes or combines service calls and hides service boundaries from clients. That directly matches the stated role. Logging records activity and a layered system describes broader organization; a generic proxy label alone does not establish the multi-service routing and composition role given in the note.

Why each choice is right or wrong

A - Incorrect: Layering organizes abstraction levels rather than naming this single microservice-facing entry component.

B - Correct: An API gateway hides service boundaries behind a unified entry point and routes or combines requests.

C - Incorrect: API logging records activity rather than routing clients across hidden service boundaries.

D - Incorrect: A generic proxy forwards requests; the note names the broader multi-service entry-point role an API gateway.

Primary note proof

Study-note section SE24

An API gateway presents one entry point and routes/combines calls to services, hiding service boundaries from clients. It can centralize policies but also become a bottleneck. A component diagram shows components/interfaces/dependencies; a deployment diagram places software artifacts on nodes. Choose the style for the required interactions and qualities rather than selecting a universally best architecture.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 119 | lines 13-25

6.1.2. Services and subsystem interfaces
A subsystem is characterized by the services it provides to other subsystems. A service is a set of related operations that share a common purpose. A subsystem providing a notification service, for
example, defines operations to send notices, look up notification channels, and subscribe and unsubscribe to a channel.
The set of operations of a subsystem that are available to other subsystems form the subsystem interface. The subsystem interface, also referred to as the application programmer interface (API),
includes the name of the operations, their parameters, their types, and their return values. System
design focuses on defining the services provided by each subsystem that is, enumerating the operations, their parameters, and their high-level behavior. Object design will focus on defining the subsystem interfaces, that is, the type of the parameters and the return value of each operation.
The definition of a subsystem in terms of the services it provides helps us focus on its interface as opposed to its implementation. A good subsystem interface should provide as little information

Software Engineering module | PDF page 124 | lines 8-16

Client/server architecture
In the client/server architecture (Figure 6.9), a subsystem, the server, provides servicesto instances
of other subsystems called the clients, which are responsible for interacting with theuser. The request for a service is usually done via a remote procedure call mechanism or acommon object broker (e.g., CORBA or Java RMI). Control flow in the clients and the servers isindependent except for synchronization to manage requests or to receive results.
Figure 6.9 Client/server architecture (UML class diagram).
Clients request services from one or more servers. The server has no knowledge of the client. The client/server architecture is a generalization of the repository architecture.

Question sources

Computer Science model exam collection · Question 44

Question 143 | SS139

Which question is outside the architectural abstraction level and instead asks for every source-level execution detail?

A. Which architectural organization is best for delivering the functional requirements?

B. How the system will be distributed across a number of cores or processors

C. Which variable is assigned at every individual source-code execution step?

D. Which architectural patterns or styles to use

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Which variable is assigned at every individual source-code execution step?

Architecture addresses major organization, responsibilities, allocation and style decisions. Selecting an arrangement for required behavior, mapping work to processors and choosing patterns are therefore valid architectural concerns. Enumerating every source-level assignment on every execution step is implementation-level tracing, beyond the abstraction of architecture documentation. Option C supplies that genuine exception while preserving the question's comparison of architectural and detailed concerns.

Why each choice is right or wrong

A - Incorrect: Organization supporting required behavior is architectural.

B - Incorrect: Processor allocation is an architectural concern.

C - Correct: Every source-level execution step exceeds the architectural abstraction.

D - Incorrect: Style and pattern selection is architectural.

Primary note proof

Study-note section SE23

> A structure is the elements/relations themselves; a view represents selected aspects. Module structures show code organization, component-and-connector structures runtime interaction, and allocation structures map software to hardware, teams or environment. Processes/threads belong in concurrency views. Evaluate architectures against quality attributes and scenarios; processor allocation, decomposition and style selection can all be architectural concerns.

Study-note section SE21

> System design turns the analysis model into architecture: design goals, subsystem responsibilities, dependencies, hardware mapping, data management, access control, control flow and startup/shutdown/error policies. Detail/object design refines APIs, parameter/return types, exceptions, component choices and algorithms. A subsystem is a manageable logical collection with services and an interface; it need not be a physical replaceable artifact.

Study-note section SE21

Architecture documentation supports communication and records major decisions, not every source-level execution step.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 116 | lines 16-29

The analysis model describes the system completely from the actors' point of view and serves as the basis of communication between the client and the developers. The analysis model, however, does not contain information about the internal structure of the system, its hardware configuration,
or, more generally, how the system should be realized. System design is the first step in this direction. System design results in the following products:
• List of design goals, describing the qualities of the system that developers should optimize
• Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.
The design goals are derived from the nonfunctional requirements. Design goals guide the decisions to be made by the developers especially when trade-offs are needed. The subsystem decomposition constitutes the bulk of system design. Developers divide the system into manageable pieces to deal with complexity: Each subsystem is assigned to a team and realized independently. In order for this to be possible, though, developers need to address system-wide

Software Engineering module | PDF page 121 | lines 6-15

6.1.4. Software architecture
As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
In this section, we describe a few sample architectures that can be used for different systems. This
is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
representative examples and refer the reader to the literature for more details.

Question sources

Computer Science model exam collection · Question 46

Question 144 | SS144

Select the one that is not related with the purpose of properly documenting software architectures

A. to Know how the system is organized and interoperate

B. For Stakeholder communication

C. For critical system design decisions

D. To easily figure out the source code flow

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: To easily figure out the source code flow

The SDD documents goals, subsystem organization, mapping and interfaces to support teams and architecture decisions. Detailed source-code control flow is a lower-level implementation concern than the document's architectural control-flow mechanisms.

Why each choice is right or wrong

A - Incorrect: Subsystem decomposition and interfaces expose organization/interoperation. See PDF p. 130.

B - Incorrect: Defines interfaces between development teams and supports communication. See PDF p. 130.

C - Incorrect: Serves as reference when architecture-level decisions are revisited. See PDF p. 130.

D - Correct: Detailed source-level flow is not the purpose of an architecture description; global control mechanisms may still be documented. See PDF p. 130.

Primary note proof

Study-note section SE21

> System design turns the analysis model into architecture: design goals, subsystem responsibilities, dependencies, hardware mapping, data management, access control, control flow and startup/shutdown/error policies. Detail/object design refines APIs, parameter/return types, exceptions, component choices and algorithms. A subsystem is a manageable logical collection with services and an interface; it need not be a physical replaceable artifact.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 130 | lines 3-11

> 6.3. DOCUMENTING SYSTEM DESIGN
> System design is documented in the System Design Document (SDD). It describes design goals set by the project, the subsystem decomposition (with UML class diagrams), the hardware/software mapping (with UML deployment diagrams), the data management, the access

control, control flow mechanisms, and boundary conditions. The SDD is used to define interfaces between teams of developers and as reference when architecture-level decisions need to be revisited. The audience for the SDD includes the project management, the system architects (i.e., the developers who participate in the system design), and the developers who design and implement each subsystem. The following is an example template for a SDD:

Question sources

Computer Science model exam collection · Question 51

Question 145 | SS147

Given the following fragment of code, how many tests are required for 100% decision coverage?
if width > length
then biggest_dimension = width
if height > width
then biggest_dimension = height
end_if
else biggest_dimension = length
if height > length
then biggest_dimension = height
end_if
end_if

A. ona. 4

B. 6

C. Ole 2

D. 3

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer A: Four tests for 100% decision coverage.

There are three decisions: width>length and one separate height comparison inside each outer branch. Covering both inner outcomes requires two tests with outer true and two with outer false. For example (w,l,h)=(3,2,4),(3,2,1),(2,3,4),(2,3,1) covers all outcomes. One execution cannot visit both outer branches, so two or three tests cannot suffice; six are unnecessary.

Why each choice is right or wrong

A - Correct: Four tests cover true/false of the inner decision on each of the two outer branches.

B - Incorrect: Six can cover the decisions, but four suffice; it is not the required minimum.

C - Incorrect: Two cover the outer decision but cannot cover both outcomes of both separate inner decisions.

D - Incorrect: Three leave at least one inner decision outcome uncovered because each outer branch needs two tests.

Primary note proof

Study-note section SE38

Statement coverage is the number of distinct executable statements exercised divided by the total executable statements in the declared test scope. Unreachable statements can prevent100%; any justified exclusions must be documented with the adjusted scope. Decision coverage counts decision outcomes; branch coverage counts control-flow edges, including unconditional edges under the stated convention. Testing both true and false of each decision may need more than two overall test cases when branches nest. Coverage is evidence of exercised structure, not proof of all behavior.

Study-note section SE38

In the width/length/height program, the outer if has separate inner decisions in each branch. Need outer true with inner true and false, and outer false with its inner true and false: four tests, for example (w,l,h)=(3,2,4),(3,2,1),(2,3,4),(2,3,1). One run cannot exercise both outer branches. All paths may be infeasible or infinite with loops; specify the chosen coverage target.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 24-29

Path testing
Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path testing is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.
The starting point for path testing is the flow graph. A flow graph consists of nodes representing

Software Engineering module | PDF page 147 | lines 3-8

• A test stub is a partial implementation of components on which the tested component depends. A test driver is a partial implementation of a component that depends on the tested component. Test stubs and drivers enable components to be isolated from the rest of the system for testing.
• A correction is a change to a component. The purpose of a correction is to repair a fault. Note that a correction can introduce new faults.

Question sources

Computer Science model exam collection · Question 54

Question 146 | SS148

Using classical up-front waterfall planning, usable incremental additions and timeboxed adaptive Agile, which lifecycle statements are NOT true? Select all that apply.

A. Ina waterfall life cycle, the project scope, time, and cost are determined in the early phases of the life cycle

B. Inagile life cycle the project scope is generally determined early in the project life cycle, but time and cost estimates are routinely modified

C. Inanincremental life cycle, the deliverable is produced through a series of iterations that successively add functionality within a predetermined time frame

D. Inagile life cycles, the project scope cannot be outlined and agreed before the start of iteration.

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B, D: Inagile life cycle the project scope is generally determined early in the project life cycle, but time and cost estimates are routinely modified; Inagile life cycles, the project scope cannot be outlined and agreed before the start of iteration.

Classical waterfall establishes planned baselines early, and incremental work adds functionality in successive deliveries. Agile instead adapts content while its timeboxes tend to remain fixed, so B misstates the general organizing pattern. D is also false because Agile permits outlining and agreeing scope before an iteration; planning continues as evidence changes. Adaptation does not mean that scope discussion or early agreement is prohibited.

Why each choice is right or wrong

A - Incorrect: Early planned baselines fit classical waterfall.

B - Correct: The claim reverses the timeboxed/adaptive-content characterization.

C - Incorrect: Successive additions of usable functionality fit incremental delivery.

D - Correct: Agile permits scope discussion and agreement before an iteration.

Primary note proof

Study-note section SE05

> Scrum uses timeboxed Sprints of at most one month, with a Sprint Goal, selected work and a usable Increment meeting a Definition of Done. Scope is discussed before and during planning; it is not forbidden to agree scope. Timeboxes tend to stay fixed while content is adapted without endangering the goal. Agile can be combined with required assurance/documentation; safety-critical context alone cannot prove it is unsuitable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
> follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Software Engineering module | PDF page 22 | lines 8-12

> This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
> also provide a platform for evaluation by the user.

Software Engineering module | PDF page 20 | lines 18-22

> the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
> In principle, the waterfall model should only be used when the requirements are well understood
> and unlikely to change radically during system development. However, the waterfall model

Question sources

Computer Science model exam collection · Question 55

Question 147 | SS151

Which design pattern is used to restore the state of an object to its previous state?

A. Iterator pattern

B. Observer pattern

C. Momento pattern

D. Visitor pattern

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Memento, printed as Momento.

SE27 explicitly identifies Memento as storing and restoring state while hiding representation, and notes that Momento in this question bank is a spelling error for Memento. That makes C the intended pattern name. Iterator traverses a collection and Observer notifies subscribers; neither supplies the state-snapshot restoration responsibility asked for here.

Why each choice is right or wrong

A - Incorrect: Iterator traverses elements rather than storing a restorable snapshot of object state.

B - Incorrect: Observer notifies subscribers about changes rather than restoring an earlier object state.

C - Correct: Momento refers here to Memento, the state capture-and-restoration pattern.

D - Incorrect: Visitor is not the state-snapshot restoration pattern; the note assigns that role to Memento.

Primary note proof

Study-note section SE27

> Factory Method lets subclasses choose a concrete product through a creation operation, reducing coupling to concrete constructors. A simple factory centralizes creation but is not identical to Factory Method. Singleton controls access to one instance; Observer notifies subscribers; Memento stores/restores state while hiding representation; Iterator traverses; Decorator adds behavior through wrapping. The pattern name is Memento, not Momento.

Study-note section SE27

> Variability can use selectable features, omission/inclusion and extension points; overloading supplies alternate signatures in code but is not every variability mechanism. Testability improves with controllable inputs, observable outputs, replaceable dependencies and limited structural complexity. A sandbox can isolate execution. Assess each design claim independently before evaluating a combined statement.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 104 | lines 6-17

> 5.1.4. Generalization
> Generalization enables us to organize concepts in to hierarchies. There may be any number of intermediate levels in between, covering more-or-less generalized concepts (e.g., Low Priority Incident, Emergency, Disaster). Such hierarchies allow us to refer to many concepts precisely. When
> we use the term Incident, we mean all instances of all types of Incidents. When we use the term Emergency, we only refer to an Incident that requires an immediate response. This view of generalization stems from modeling.
> In an object-oriented programming language, inheritance is a reusability technique. If a class Child
> inherits from a class Parent, all the attributes and methods available on Parent are automatically available on Child. The class Child may add additional methods or override inherited methods, thus refining the class Parent. As in the case of generalization, the class at the top of the hierarchy
> tends to be the most general one, whereas the leaves tend to be the most specialized.

Software Engineering module | PDF page 121 | lines 6-15

6.1.4. Software architecture
As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
In this section, we describe a few sample architectures that can be used for different systems. This
is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
representative examples and refer the reader to the literature for more details.

Question sources

Computer Science model exam collection · Question 58

Question 148 | SS153

Which of the following shows the five stages in Tuckman's model of team development, in sequential order?

A. forming, storming, performing, norming, and adjourning

B. norming, forming, storming, performing, and adjourning

C. storming, forming, norming, performing, and adjourning

D. forming, storming, norming, performing, and adjourning

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer D: Forming → storming → norming → performing → adjourning.

SE32 lists orientation first, then conflict, development of norms, effective performance and closure. Their stage names are forming, storming, norming, performing and adjourning. D preserves that order. The other alternatives either put performance before norms or move orientation after later stages, contradicting the taught sequence.

Why each choice is right or wrong

A - Incorrect: It places performing before norming, reversing the stated development sequence.

B - Incorrect: It starts with norming before the initial forming and storming stages.

C - Incorrect: It places storming before the initial forming stage.

D - Correct: It matches the taught progression from orientation through conflict, norms, performance and closure.

Primary note proof

Study-note section SE32

Functional organizations group specialists by discipline; project organizations group people around projects. Functional specialization helps staffing and knowledge continuity but may create handoff barriers. Democratic teams share decisions; chief-programmer teams centralize technical coordination. These are different classification dimensions. Tuckman's sequence is forming→storming→norming→performing→adjourning: orientation, conflict, norms, effective work and closure.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 96 | lines 3-11

Team Structure
• Team structure addresses the issue of organization of the individual project teams. There are some possible ways in which the individual project teams can be organized.
• Problems of different complexities and sizes often require different team structures for chief solution.
• The most common types of team structure are:
· Chief Programmer Team
· Democratic Team
· Mixed Control Team Organization

Question sources

Computer Science model exam collection · Question 60

Question 149 | SS154

Which one of the following is not among the purposes of software testing?

A. Requesting more design and implementation time

B. Identifying shortcomings

C. Improving product acceptance

D. Enhancing reliability

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer A: Requesting more design and implementation time

Testing detects faults, supports client acceptance and enables corrections that improve reliability. Requesting more implementation time is a possible project consequence, not a testing purpose.

Why each choice is right or wrong

A - Correct: A request for time is not the fault-detection or confidence purpose. See PDF p. 140, 140, 145.

B - Incorrect: Identifying faults/shortcomings is the explicit goal. See PDF p. 140, 140, 145.

C - Incorrect: Client acceptance is checked through acceptance tests. See PDF p. 140, 140, 145.

D - Incorrect: Fault discovery permits corrections and increased reliability. See PDF p. 140, 140, 145.

Primary note proof

Study-note section SE34

> A fault/defect is an underlying problem; an error may be an incorrect internal state; a failure is an externally observed departure from required service. Reliability is a product quality attribute; low cost or short development time are project outcomes. Tests increase confidence and reveal defects but cannot generally guarantee correctness over every possible input.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 140 | lines 27-32

> 7.1 OVERVIEW OF SOFTWARE QUALITY ASSURANCE
> We define testing as the systematic attempt to find errors in a planned way in the implemented software. Contrast this definition with another commonly used definition that says that “testing is
> the process of demonstrating that errors are not present.” The distinction between these two definitions is important. Our definition does not mean we simply demonstrate that the program does what it is intended to do. The explicit goal of testing is to demonstrate the presence of faults.

Software Engineering module | PDF page 140 | lines 11-18

> Reliability is a measure of success with which the observed behavior of a system conforms to the specification of its behavior. Software reliability is the probability that a software system will not cause the failure of the system for a specified time under specified conditions. Failure is any deviation of the observed behavior from the specified behavior. An error means the system is in a state such that further processing by the system will lead to a failure, which causes the system to
> deviate from its intended behavior. A fault, also called defect or bug, is the mechanical or algorithmic cause of an error. The goal of testing is to maximize the number of discovered faults, which then allows developers to correct them and increase the reliability of the system.

Software Engineering module | PDF page 145 | lines 3-5

> o Acceptance testing and installation testing check the requirements against the
> project agreement and should be done by the client, if necessary with support by
> the developers.

Question sources

Computer Science model exam collection · Question 61

Question 150 | SS155

During which listed activity should the test plan define configuration-management procedures alongside objectives, resources and schedule, before those procedures are applied during execution?

A. During test execution

B. During test planning

C. During test closing

D. During test initiation

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: During test planning

Test planning specifies intended work, objectives, resources and schedule; configuration-management procedures should be defined early as part of that preparation. They then apply throughout testing to preserve reproducible builds, data and environments. Execution and closing use those procedures rather than being their exclusive establishment point. Initiation may set context, but the question specifically asks for the activity that defines them in the test plan.

Why each choice is right or wrong

A - Incorrect: Execution applies the planned procedures to tested versions.

B - Correct: Test planning defines procedures with the other planned work.

C - Incorrect: Closing preserves final artifacts under established control.

D - Incorrect: Initiation sets context rather than the specified test-plan definition activity.

Primary note proof

Study-note section SE33

> Configuration management identifies controlled items, versions/baselines, change procedures, status and audits. Items may include source code, requirements, manuals, test plans, environments and build configurations. A team’s ordinary browser is usually a tool rather than product CI, but a controlled test-browser version may be an environment CI.

Study-note section SE39

> A test plan describes intended work; incident/defect reports are normally execution outputs. A defect report should identify build/environment, steps, expected versus actual behavior, evidence and severity. Anyone finding a fault should record it; testers often coordinate

reporting, but it is not their exclusive permission. Configuration control makes the tested version reproducible.

Study-note section SE33

Plan test configuration management early and apply it throughout testing, not at just one exclusive phase.

Study-note section SE39

Analysis asks what to test and assesses testability; design asks how.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 5-10

Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
faults if change is controlled.

Software Engineering module | PDF page 90 | lines 7-24

4.2. Project Planning
• Probably the most time-consuming project management activity.
• Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
• Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
The project plan sets out:
• The resources available to the project;
• The work breakdown;
• A schedule for the work.
Project Plan Structure
• Introduction.
• Project organisation.
• Risk analysis.
• Hardware and software resource requirements.
• Work breakdown.
• Project schedule.
• Monitoring and reporting mechanisms.

Question sources

Computer Science model exam collection · Question 62

Question 151 | SS156

"The system architecture should be designed using fine-grain, self-contained components. Producers of data should be separated from consumers
and shared data structures should be avoided". For which requirement this architecture description applies?

A. Performance

B. Availability

C. Maintainability

D. Security

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Maintainability

Independent self-contained subsystems and avoiding shared attribute data reduce coupling, limiting the impact of future changes. That directly serves maintainability.

Why each choice is right or wrong

A - Incorrect: Fine-grained separation can carry overhead and is not primarily the stated speed optimization. See PDF p. 120, 118.

B - Incorrect: Availability requires continuity/failure treatment beyond component separation alone. See PDF p. 120, 118.

C - Correct: Matches minimizing change impact through low coupling. See PDF p. 120, 118.

D - Incorrect: Security needs explicit access-control policy; independence alone does not establish it. See PDF p. 120, 118.

Primary note proof

Study-note section SE43

> Choose designs by their effect on stated goals. Fine-grained self-contained components, separation of producers/consumers and limited shared mutable data often improve maintainability. Trade-offs remain: interfaces can add latency, isolation can complicate transactions, and global state can simplify one operation while coupling many others. Prefer measurable quality scenarios to counting components alone.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 120 | lines 6-12

> 6.1.3. Coupling and coherence
> Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will

have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Software Engineering module | PDF page 118 | lines 11-16

6.1.1. Subsystems and classes
In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Question sources

Computer Science model exam collection · Question 63

Question 152 | SS159

Which of the following is a process of selecting program paths in such a manner that certain branches (i.e., outgoing edges of nodes) of a control flow
graph are covered by the execution of those paths?

A. Branch boundary

B. Branch coverage

C. Branch control

D. Branch expansion

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: Branch coverage.

The stem explicitly defines branches as outgoing control-flow edges. SE38 identifies branch coverage as counting exercised control-flow edges under the declared convention. This differs from merely counting executed statements or evaluating decision outcomes. Branch boundary, branch control and branch expansion are not the coverage criterion described by selecting paths that exercise graph edges.

Why each choice is right or wrong

A - Incorrect: Branch boundary is not the defined edge-coverage criterion in the note.

B - Correct: Branch coverage measures whether control-flow edges are exercised by the selected test paths.

C - Incorrect: Branch control is not the name of the specified structural coverage measure.

D - Incorrect: Branch expansion does not name the outgoing-edge coverage criterion described.

Primary note proof

Study-note section SE38

> Statement coverage is the number of distinct executable statements exercised divided by the total executable statements in the declared test scope. Unreachable statements can prevent100%; any justified exclusions must be documented with the adjusted scope. Decision coverage counts decision outcomes; branch coverage counts control-flow edges, including unconditional edges under the stated convention. Testing both true and false of each decision may need more than two overall test cases when branches nest. Coverage is evidence of exercised structure, not proof of all behavior.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 24-29

> Path testing
> Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path testing is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.
> The starting point for path testing is the flow graph. A flow graph consists of nodes representing

Question sources

Computer Science model exam collection · Question 66

Question 153 | SS161

Which of the following is the correct sequence that will be followed during software evolution process?

A. Change request, release planning, change implementation, impact analysis, system release

B. Change request, impact analysis, release planning, change implementation, system release

C. Impact analysis, Change request, change implementation, release planning, system release

D. Change request, impact analysis, change implementation, release planning, system release

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer B: Change request → impact analysis → release planning → implementation → release.

SE40 gives the sequence directly. A requested change is analyzed before commitment to a release, then planned, implemented and tested, and finally released. B preserves those dependencies. The alternatives analyze consequences too late, begin analysis before a request exists, or implement before release planning. Implementation here includes the testing work omitted from the abbreviated options.

Why each choice is right or wrong

A - Incorrect: It postpones impact analysis until after implementation, too late to guide planning and commitment.

B - Correct: It preserves request, analysis, planning, implementation/testing and release in the taught order.

C - Incorrect: It begins impact analysis before identifying a change request and implements before release planning.

D - Incorrect: It places implementation before release planning instead of planning the analyzed change first.

Primary note proof

Study-note section SE40

> A typical evolution sequence is change request→impact analysis→release planning→implementation/testing→release. Among a list lacking the request step, impact analysis may be the earliest listed technical activity. Major restructuring is reengineering; if the architecture itself changes, architectural transformation is the more specific description. Markets, technology, stakeholders and funding can all change scope or priorities.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 41 | lines 16-17

> ✓ Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.

Software Engineering module | PDF page 19 | lines 24-29

> 5. Operation and maintenance: Normally (although not necessarily), this is the longest life cycle phase. Maintenance of a typical software product requires much more than the effort necessary to
> develop the product itself. The system is installed and put into practical use. Maintenance involves
> correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system ‘s services as new requirements are discovered.

Question sources

Computer Science model exam collection · Question 68

Question 154 | SS162

Which of the following UML element is wrongly applied?

A. Use case diagram represents an interaction with the system

B. Sequence diagram shows the sequence of interactions required to complete some operation

C. An activity diagram primarily specifies the persistent lifecycle states of one object and the event-triggered transitions between those states.

D. Class diagram shows the object classes in a system and their relation

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: An activity diagram primarily specifies the persistent lifecycle states of one object and the event-triggered transitions between those states.

Use-case diagrams summarize actor goals, sequence diagrams order interactions across participants, and class diagrams describe static types and relationships. An activity diagram models workflow and action/control flow. Persistent lifecycle states and event-triggered transitions of a particular object instead call for a state machine. The revised third statement makes that mismatch explicit; the original generic event wording was too broad because activities can also respond to events.

Why each choice is right or wrong

A - Incorrect: Use-case diagrams summarize external roles and system goals/interactions.

B - Incorrect: Sequence diagrams show ordered participant interactions.

C - Correct: The specified lifecycle-state behavior belongs to a state machine rather than an activity workflow.

D - Incorrect: Class diagrams describe static classes and their relationships.

Primary note proof

Study-note section SE20

> An activity diagram models workflow: filled circle start, rounded actions, guarded decision diamond, thick fork/join bars and partitions/swimlanes for responsibility. A diamond can also merge alternative flows; a join synchronizes concurrent flows. A state machine models states and transitions triggered by events, often with guards/actions, for one object's lifecycle.

Study-note section SE19

> Sequence diagrams show interactions across lifelines, with order/time generally downward. Messages cross between participants; activation rectangles show execution. A creation message starts a lifeline; an X marks modeled destruction, not a guarantee about when a garbage

collector runs. Synchronous calls wait for completion; asynchronous messages permit the sender to proceed.

Study-note section SE17

A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
Use cases describe the behavior of the system as seen from an actor's point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
use case to access the system functionality. The use case can then initiate other use cases and gather
more information from the actors. When actors and use cases exchange information, they are said
to communicate. We will see later that we represent these exchanges with communication relationships.

Software Engineering module | PDF page 54 | lines 26-35

both buttons simultaneously. Figure 2.10 depicts a sequence diagram for an actor setting his 2Bwatch one minute ahead.
Each column represents an object that participates in the interaction. The vertical axis represents
time from top to bottom. Messages are shown by arrows. Labels on arrows represent message names and may contain arguments. Activations (i.e., executing methods) are depicted by vertical rectangles. Actors are shown as the leftmost column.
Sequence diagrams can be used to describe either an abstract sequence (i.e., all possible interactions) or concrete sequences (i.e., one possible interaction, as in Figure 2.10). When describing all possible interactions, sequence diagrams also provide notations for conditionals and
iterators.

Software Engineering module | PDF page 56 | lines 18-24

2.3.4. Activity diagrams
The outgoing transitions are triggered by the completion of an action associated with the state. This is called an action state. By convention, the name of a state denotes a condition, whereas the name of an action state denotes an action. Activity diagrams are statechart diagrams whose states
are action states. is an activity diagram corresponding to the state diagram. An alternate and

equivalent view of activity diagrams is to interpret action states as control flow between activities
and transitions; that is, the arrows are interpreted as sequential constraints between activities.

Software Engineering module | PDF page 55 | lines 4-9

2.3.3. State chart Diagram
A UML statechart is a notation for describing the sequence of states an object goes through in response to external events. Statecharts are extensions of the finite state machine model. On the one hand, statecharts provide notation for nesting states and state machines (i.e., a state can be described by a state machine). On the other hand, statecharts provide notation for binding transitions with message sends and conditions on objects.

Software Engineering module | PDF page 51 | lines 14-24

2.3.2. Class Diagrams
Classes and objects
Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behavior of a set of objects. Objects are entities that encapsulate state and behavior. Each object has an identity: It can be referred individually and is distinguishable from other objects.
In UML, classes and objects are depicted by boxes including three compartments. The top compartment displays the name of the class or object. The center compartment displays its attributes, and the bottom compartment displays its operations. The attribute and operation compartment can be omitted for clarity. Object names are underlined to indicate that they are instances. By convention, class names start with an uppercase letter. Objects in object diagrams

Question sources

Computer Science model exam collection · Question 69

Question 155 | SS163

Select the wrong statement regarding traceability in software development projects.

A. Traceability shows the dependency among two or more requirements if any

B. Requirements should be linked to the respective stakeholder who generated them

D. Every design element(component) should be linked back to the requirement

G. Traceability traces the overall cost and schedule of every activity in the requirement

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer G: Traceability traces the overall cost and schedule of every activity in the requirement

Traceability records where requirements came from, how requirements depend on one another, and how each requirement is realized and checked in design, code and tests. Source links support rationale, and forward or backward links support change-impact analysis. A cost-and-schedule ledger instead belongs to project planning and control. It can reference requirements, but recording every activity's cost and schedule is not the definition of requirements traceability.

Why each choice is right or wrong

A - Incorrect: Dependency links among requirements are a valid traceability relation.

B - Incorrect: Source or stakeholder links record a requirement's provenance.

D - Incorrect: Design-to-requirement links support realization and impact analysis.

G - Correct: An overall activity cost-and-schedule ledger is project control, not the defining traceability relation.

Primary note proof

Study-note section SE11

> Trace each requirement backward to its source/rationale and forward to design, code and tests. Record dependencies among requirements. A traceability matrix links these artifacts and helps impact analysis; it is not just a cost/schedule ledger. Stakeholders must negotiate conflicting requirements rather than independently approving incompatible demands. Scope, understanding and volatility are distinct elicitation difficulties.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 70 | lines 3-16

> 3.2.5. Verifiability and Traceability
> Two more desirable properties of a system specification are that it be verifiable and traceable. The
> specification is verifiable if, once the system is built; a repeatable test can be designed to demonstrate that the system fulfills the requirement. For example, a mean time to failure of a hundred years for SatWatch would be difficult to achieve (assuming it is realistic in the first place).
> The following requirements are additional examples of non-verifiable requirements:
> • The product shall have a good user interface (good is not defined).
> • The product shall be error free (requires large amount of resources to establish).
> • The product shall respond to the user with 1 second for most cases ("most cases" is not defined).
> A system specification is traceable if each system function can be traced to its corresponding set of requirements. Traceability is not a constraint on the content of the specification, but rather, on its organization. Traceability facilitates the development of tests and the systematic validation of the design against the requirements.

Software Engineering module | PDF page 90 | lines 7-24

4.2. Project Planning
• Probably the most time-consuming project management activity.
• Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
• Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
The project plan sets out:
• The resources available to the project;
• The work breakdown;
• A schedule for the work.
Project Plan Structure
• Introduction.
• Project organisation.
• Risk analysis.
• Hardware and software resource requirements.
• Work breakdown.
• Project schedule.
• Monitoring and reporting mechanisms.

Question sources

Computer Science model exam collection · Question 70

Question 156 | SS164

Which named prioritization heuristic suggests that a small subset of causes may account for much of the observed impact, often summarized as 80/20?

A. Pareto principle

B. Pairwise principle

C. Partition principle

D. Parametric principle

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer A: Pareto principle

Pareto's 80/20 idea is a prioritization heuristic: a relatively small set of causes may account for a large share of observed impact. It encourages examining evidence to focus improvement effort. It does not promise that exactly one fifth of the effort fixes exactly four fifths of all defects. The revised question asks for this named heuristic, so the other principle names do not identify the rule described.

Why each choice is right or wrong

A - Correct: Pareto names the small-subset/large-impact heuristic often expressed as 80/20.

B - Incorrect: This is not the name of the described 80/20 prioritization heuristic.

C - Incorrect: Partitioning does not name the described concentration-of-impact heuristic.

D - Incorrect: Parametric reasoning does not name the described concentration-of-impact heuristic.

Primary note proof

Study-note section SE43

> Pareto's 80/20 rule is a prioritization heuristic: a small subset of causes may account for much of the observed impact. It does not guarantee 80% of defects can be fixed with exactly 20% effort. Refactor to preserve behavior while improving structure; optimize only against measured needs; record rationale and update traceability and regression evidence after change.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 91 | lines 37-42

> • Software Metrics can be used to indicate the basic attributes of the software. The indicative metrics, not exhaustive, are
> • Size - Line of code ,Function point count
> • Quality- Reliability, Accuracy, Dependability, and these are measured through some measures like errors, failures and number of runs between failures ,etc.
> • Execution Time -process time of execution of a critical process and many more.

Question sources

Computer Science model exam collection · Question 71

Question 157 | SS166

A business expects changing requirements and organizes development around repeated customer collaboration and incremental replanning; risk-analysis cycles are not its organizing priority. Which offered approach most directly matches these priorities?

A. Code and run

B. Spiral model

C. Agile development method

D. Water fall model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Agile development method

The scenario's organizing emphasis is repeated customer collaboration and adaptive planning as needs change, which identifies Agile. Spiral can also accommodate change, but organizes its cycles around explicit risk assessment and reduction. Classical waterfall relies more heavily on stable requirements and planned phase commitments. Code-and-run lacks the described disciplined planning and feedback organization. Changing requirements alone would not uniquely determine a process, so the added priorities matter.

Why each choice is right or wrong

A - Incorrect: Code-and-run does not supply the disciplined adaptive planning and feedback organization stated.

B - Incorrect: Spiral is specifically organized around risk assessment and reduction rather than the stated primary emphasis.

C - Correct: Agile makes customer collaboration and adaptation central to planning.

D - Incorrect: Classical waterfall is a poorer fit for repeated replanning of evolving needs.

Primary note proof

Study-note section SE05

> Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
> follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Software Engineering module | PDF page 20 | lines 18-22

> the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to

respond to changing customer requirements.
In principle, the waterfall model should only be used when the requirements are well understood
and unlikely to change radically during system development. However, the waterfall model

Question sources

Computer Science model exam collection · Question 73

Question 158 | SS168

Which statement is correct about the differences between functional and non-functional requirements?

A. Functional requirements are the services the system provide while non-functional requirements are constraints on the services or functions offered by the system

B. Functional requirements are decided by the customer while non-functional requirements are decided by developers

C. Both are the same most of the time and only vary under certain contexts

D. Unlike functional requirement, nonfunctional requirements are stable

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: Functional requirements are the services the system provide while non-functional requirements are constraints on the services or functions offered by the system

Required interactions are distinct from user-visible quality constraints such as response time and accuracy; users and clients participate in defining both.

Why each choice is right or wrong

A - Correct: Correct service-versus-constraint distinction. See PDF p. 65, 66, 78.

B - Incorrect: Both types involve users/clients and developers; they are not separated by who decides. See PDF p. 65, 66, 78.

C - Incorrect: The module explicitly distinguishes their subject matter. See PDF p. 65, 66, 78.

D - Incorrect: No immunity to changing stakeholder needs is stated for nonfunctional requirements. See PDF p. 65, 66, 78.

Primary note proof

Study-note section SE10

Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning. Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 65 | lines 19-23

3.2.1. Functional Requirements
Functional requirements describe the interactions between the system and its environment independent of its implementation. The environment includes the user and any other external system with which the system interacts. For example, the following is an example of functional requirements for SatWatch, a watch that resets itself without user intervention:

Software Engineering module | PDF page 66 | lines 14-18

3.2.2. Non-Functional and Pseudo-Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related with the functional behavior of the system. Nonfunctional requirements include quantitative constraints, such as response time (i.e., how fast the system reacts to user commands)
or accuracy (i.e., how precise are the system's numerical answers). The following are the

Software Engineering module | PDF page 78 | lines 12-30

3.3.7. Identifying Non-Functional Requirements
Nonfunctional requirements describe user-visible aspects of the system that are not directly related
to the functional behavior of the system. Nonfunctional requirements span a number of issues, from user interface look and feel to response time requirements to security issues. Nonfunctional
requirements are defined at the same time as functional requirements are, because they have as much impact on the development and cost of the system.
For example, consider a mosaic display that an air traffic controller uses to track planes. A mosaic
display system complies data from a series of radar and databases (hence the term "mosaic") into
a summary display indicating all aircraft in a certain area, including their identification, speed, and
altitude. The number of aircraft such a system can display constrains the performance of the air traffic controller and the cost of the system. If the system can only handle a few aircraft simultaneously, the system cannot be used at busy airports. On the other hand, a system able to handle a large number of aircraft is more costly and more complex to build.
Nonfunctional requirements can be elicited by investigating the following issues.
• User interface and human factors. What kind of interface should the system provide? What is the level of expertise of the users?
• Documentation. What level of document is required? Should only user documentation be

provided? Should there be technical documentation for maintainers? Should the development process be documented?

Question sources

Computer Science model exam collection · Question 75

Question 159 | SS169

Which paired activity assesses requirements/system testability and then designs test cases and expected results?

A. Test analysis and requirements

B. Test analysis and design

C. Test analysis and planning

D. Test analysis and Implementation

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: Test analysis and design

Test analysis examines the test basis, identifies what can be tested and evaluates testability. Test design then specifies how to test it through cases and expected results. The analysis-and-design pair therefore covers both responsibilities. Requirements work, planning and implementation are relevant elsewhere, but do not replace case design: planning addresses resources and scope, while implementation prepares data, scripts and the environment.

Why each choice is right or wrong

A - Incorrect: Requirements work does not supply the requested case-design half.

B - Correct: Analysis assesses testability and design specifies cases and expected results.

C - Incorrect: Planning addresses intended work, resources and schedule rather than case design.

D - Incorrect: Implementation prepares scripts, data and environment rather than being the design half.

Primary note proof

Study-note section SE39

A test plan describes intended work; incident/defect reports are normally execution outputs. A defect report should identify build/environment, steps, expected versus actual behavior, evidence and severity. Anyone finding a fault should record it; testers often coordinate reporting, but it is not their exclusive permission. Configuration control makes the tested version reproducible.

Study-note section SE39

> Analysis asks what to test and assesses testability; design asks how.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 147 | lines 14-18

> • Unit testing, which finds faults by isolating an individual component using test stubs and drivers and by exercising the component using a test case
> • Integration testing, which finds faults by integrating several components together
> • System testing, which focuses on the complete system, its functional and nonfunctional requirements, and its target environment

Question sources

Computer Science model exam collection · Question 76

Question 160 | SS174

Which parameter is used to assess and evaluate software architectures?

A. Number of components in the architecture

B. Durability of the architecture

C. Responsiveness of the architectures

D. Architectural quality attributes

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: Architectural quality attributes

Design goals are qualities derived from nonfunctional requirements; choices are evaluated consistently against performance, dependability, maintenance, cost and end-user criteria.

Why each choice is right or wrong

A - Incorrect: Component count alone does not establish architecture quality. See PDF p. 127, 116.

B - Incorrect: Durability alone does not span the taught criteria. See PDF p. 127, 116.

C - Incorrect: Responsiveness is only one performance criterion. See PDF p. 127, 116.

D - Correct: Covers the full set of quality criteria used to judge architectural decisions. See PDF p. 127, 116.

Primary note proof

Study-note section SE23

A layer is a logical abstraction level; a tier is a deployment separation. A closed layered architecture permits calls only to the immediately lower layer; an open one permits approved skipping. For top-to-bottom A,B,C,D, A→B obeys closed layering, A→C skips, and lower-to-upper calls violate the simple dependency rule unless explicitly mediated by a defined mechanism.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 127 | lines 10-28

6.2.1. Identifying design goals
The definition of design goals is the first step of system design. It identifies the qualities that our system should focus on. Many design goals can be inferred from the nonfunctional requirements or from the application domain. Others will have to be elicited from the client. It is, however, necessary to state them explicitly such that every important design decision can be made consistently following the same set of criteria.
For example, in the light of the nonfunctional requirements for MyTrip, we identify reliability and
fault tolerance to connectivity loss as design goals. We then identify security as a design goal, as numerous drivers will have access to the same trip planning server. We add modifiability as a design goal, as we want to provide the ability for drivers to select a trip planning service of their choice. The following box summarizes the design goals we identified.
In general, we can select design goals from a long list of highly desirable qualities. Tables 5-1-5-5 list a number of possible design criteria. These criteria are organized into five groups: performance, dependability, cost, maintenance, and end user criteria. Performance, dependability,
and end user criteria are usually specified in the requirements or inferred from the application domain. Cost and maintenance criteria are dictated by the customer and the supplier.
Performance criteria (Table 5-1) include the speed and space requirements imposed on the system. Should the system be responsive or should it accomplish a maximum number of tasks? Is memory space available for speed optimizations or should memory be used sparingly?

Software Engineering module | PDF page 116 | lines 16-29

The analysis model describes the system completely from the actors' point of view and serves as the basis of communication between the client and the developers. The analysis model, however, does not contain information about the internal structure of the system, its hardware configuration,
or, more generally, how the system should be realized. System design is the first step in this direction. System design results in the following products:
• List of design goals, describing the qualities of the system that developers should optimize
• Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.
The design goals are derived from the nonfunctional requirements. Design goals guide the decisions to be made by the developers especially when trade-offs are needed. The subsystem decomposition constitutes the bulk of system design. Developers divide the system into manageable pieces to deal with complexity: Each subsystem is assigned to a team and realized independently. In order for this to be possible, though, developers need to address system-wide

Question sources

Computer Science model exam collection · Question 81

Question 161 | SS176

In software risk management, accepting risk happening but don not do anything is known as:

A. Risk transfer

B. Risk Retention

C. Risk Reduction

D. Risk avoidance

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer B: Risk retention, specifically passive acceptance.

SE31 distinguishes removing a risk, reducing it, transferring consequences and retaining or accepting it. It explicitly says passive acceptance may take no action. The stem describes recognizing the risk and allowing it to remain without action, so retention matches. The other responses all involve a deliberate change to the risk or its consequences.

Why each choice is right or wrong

A - Incorrect: Transfer assigns consequences contractually rather than simply keeping the risk without action.

B - Correct: Retention or passive acceptance keeps the risk and may involve no immediate action.

C - Incorrect: Reduction actively lowers likelihood or impact instead of merely accepting exposure.

D - Incorrect: Avoidance changes the plan to remove the risk rather than allowing it to remain.

Primary note proof

Study-note section SE31

> Avoidance changes the plan to remove a risk, mitigation reduces likelihood/impact, transfer assigns consequences contractually, and retention/acceptance keeps the risk; passive acceptance may take no action. Resource leveling adjusts dates to respect resource availability and can change project finish; smoothing uses available float within schedule constraints. Neither magically creates extra staff.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 96 | lines 12-21

4.7. Risk Management
• Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
• A risk is a probability that some adverse circumstance will occur
- Project risks affect schedule or resources;
- Product risks affect the quality or performance of the software being developed;
- Business risks affect the organisation developing or procuring the software.
Risk Management Process
- Risk identification- Identify project, product and business risks;
- Risk analysis -Assess the likelihood and consequences of these risks;

Software Engineering module | PDF page 97 | lines 3-4

- Risk planning- Draw up plans to avoid or minimise the effects of the risk;
- Risk monitoring- Monitor the risks throughout the project.

Question sources

Computer Science model exam collection · Question 83

Question 162 | SS182

Which software process model will you use if you want to deliver different functionalities (modules) of the software product that have different priority
at different time?

A. Waterfall model

B. Spiral model

C. Incremental model

D. Linear model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Incremental model

Incremental development delivers prioritized functionality in usable stages. Earlier increments provide value while later increments add further capability. This is a delivery organization, distinct from a strict single-pass waterfall sequence. Spiral cycles are organized around risks and may use increments, but risk-driven development is not the specific name of the stated delivery strategy. A generic linear sequence also does not describe prioritized usable subsets.

Why each choice is right or wrong

A - Incorrect: Classical waterfall organizes phases before the principal usable delivery.

B - Incorrect: Spiral is defined by risk-driven cycles, even when an increment is produced.

C - Correct: Incremental development names the successive delivery of usable functionality.

D - Incorrect: A linear model does not name prioritized delivery of separate usable subsets.

Primary note proof

Study-note section SE04

> Choose incremental delivery when prioritized partial capability must reach users early. Smaller changes can be easier to debug and permit feedback, though architecture and integration still require planning. Iteration need not rebuild the whole system every time, and increments need not be perfectly independent modules. Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 22 | lines 8-12

> This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
> also provide a platform for evaluation by the user.

Software Engineering module | PDF page 22 | lines 24-29

> Advantages of Incremental Model:
> • Generates working software quickly and early during the software life cycle. This model is more flexible - less costly to change scope and requirements. It is easier to test and debug during a smaller iteration.
> • In this model customer can respond to each built. Lowers initial delivery cost.
> • Easier to manage risk because risky pieces are identified and handled during its iteration.

Question sources

Computer Science model exam collection · Question 89

Question 163 | SS184

Which of the following is a resource optimization techniques in which start and finish dates are adjusted based on resource constraints with the goal
of balancing the demand for resources with the available supply?

A. Resource smoothing

B. Responsibility assignment matrix

C. Resource leveling

D. Resource grouping

Hint

Write out the supplied quantities and dependency constraints. Compute and compare values under the stated metric, finish condition and assumptions.

Answer C: Resource leveling.

SE31 defines resource leveling as adjusting dates to respect resource availability, potentially changing the project finish. That matches balancing resource demand against supply by moving starts and finishes. Smoothing instead works within available float and schedule constraints. A responsibility matrix assigns ownership, and resource grouping is not the defined date-adjustment technique.

Why each choice is right or wrong

A - Incorrect: Smoothing uses available float while retaining schedule constraints rather than generally leveling against availability.

B - Incorrect: A responsibility assignment matrix maps work to owners rather than adjusting task dates for resource capacity.

C - Correct: Resource leveling changes dates to balance required resources against available supply.

D - Incorrect: Resource grouping is not the defined schedule-adjustment method described by the stem.

Primary note proof

Study-note section SE31

> Avoidance changes the plan to remove a risk, mitigation reduces likelihood/impact, transfer assigns consequences contractually, and retention/acceptance keeps the risk; passive acceptance may take no action. Resource leveling adjusts dates to respect resource availability and can change project finish; smoothing uses available float within schedule constraints. Neither magically creates extra staff.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 95 | lines 3-12

4.6. Scheduling, Organization and Team Structures
Project-task scheduling is an important project planning activity. It involves deciding which tasks would
be taken up when. In order to schedule the project activities, a software project manager needs to do
the following:
• Identify all the tasks needed to complete the project and Break down large tasks into small activities.
• Determine the dependency among different activities and Establish the most likely estimates for the time durations necessary to complete the activities and Allocate resources to activities.
• Plan the starting and ending dates for various activities and Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

Question sources

Computer Science model exam collection · Question 91

Question 164 | SS190

Identify the design principle that does not apply to software systems

A. design should be structured to accommodate change

B. design should exhibit uniformity and integration

C. design should be reinventing the wheel from scratch

D. design should be traceable to the analysis model

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: design should be reinventing the wheel from scratch

Reusable classes, controlled change and traceability to requirements support software engineering. Recreating existing solutions from scratch contradicts its reuse rationale.

Why each choice is right or wrong

A - Incorrect: Low coupling explicitly limits future change impact. See PDF p. 32, 120, 70.

B - Incorrect: Consistent interfaces and integrated structures support the system, though this exact slogan is not quoted. See PDF p. 32, 120, 70.

C - Correct: Contradicts the documented promotion of reuse and component selection. See PDF p. 32, 120, 70.

D - Incorrect: Traceability explicitly supports validating design against requirements. See PDF p. 32, 120, 70.

Primary note proof

Study-note section SE25

Use abstraction and decomposition to manage complexity; preserve consistency, traceability and clear responsibilities. Top-down design refines high-level structure; bottom-up design assembles existing components. A hybrid combines both. SOLID includes Single Responsibility, Open-Closed, Liskov substitution, Interface Segregation and Dependency Inversion. Depend on abstractions; add compatible extensions; avoid forcing unused operations. DRY limits duplicated knowledge, KISS avoids unnecessary complexity, and YAGNI avoids speculative work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 32 | lines 12-21

A class in an object-oriented system carefully delineates between its interfaces the routines and attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact
on other classes, and so, the impact is minimized. However, the object-oriented approach is not a panacea; nothing is magical here that will promote perfect design or perfect code.
Promotion of reusability
Objects are reusable because they are modeled directly out of a real-world problem domain. Each
object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.

Software Engineering module | PDF page 120 | lines 6-12

6.1.3. Coupling and coherence
Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Software Engineering module | PDF page 70 | lines 3-16

3.2.5. Verifiability and Traceability
Two more desirable properties of a system specification are that it be verifiable and traceable. The
specification is verifiable if, once the system is built; a repeatable test can be designed to demonstrate that the system fulfills the requirement. For example, a mean time to failure of a hundred years for SatWatch would be difficult to achieve (assuming it is realistic in the first place).
The following requirements are additional examples of non-verifiable requirements:
• The product shall have a good user interface (good is not defined).
• The product shall be error free (requires large amount of resources to establish).
• The product shall respond to the user with 1 second for most cases (“most cases” is not defined).

A system specification is traceable if each system function can be traced to its corresponding set of requirements. Traceability is not a constraint on the content of the specification, but rather, on its organization. Traceability facilitates the development of tests and the systematic validation of the design against the requirements.

Question sources

Computer Science model exam collection · Question 97

Question 165 | AM0001

Which phase of the software development life cycle is responsible for gathering and analyzing user requirements?

A. Analysis

B. Design

C. Development

D. Testing

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: Analysis

Requirements elicitation and analysis establish the system's purpose with users and formalize their needs; design and implementation follow.

Why each choice is right or wrong

A - Correct: The offered phase covering gathering and analysis. See PDF p. 62, 100.

B - Incorrect: Derives internal structure from requirements. See PDF p. 62, 100.

C - Incorrect: Produces executable code. See PDF p. 62, 100.

D - Incorrect: Detects faults in implemented behavior. See PDF p. 62, 100.

Primary note proof

Study-note section SE08

Business requirements explain why the project exists and its goals. User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements. A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 62 | lines 9-16

> Requirement’s elicitation focuses on describing the purpose of the system. The client, the developers, and the users identify a problem area and define a system that addresses the problem.
> Such a definition is called a system specification and serves as a contract between the client and the developers. The system specification is structured and formalized during analysis to produce an analysis model (see Figure 3.1). Both system specification and analysis model represent the same information. They differ only in the language and notation they use. The system specification
> is written in natural language, whereas the analysis model is usually expressed in a formal or semiformal notation.

Software Engineering module | PDF page 100 | lines 11-24

> Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users
> Figure 4.1 Products of requirements elicitation and analysis (UML activity diagram).
> Althoughtheanalysismodelmaynotbeunderstandabletotheusersandtheclient, it helps developers verify the system specification produced during requirements elicitation. There is a natural tendency for users and developers to postpone difficult decisions until later in the project. A decision may be difficult because of lack of domain knowledge, lack of technological knowledge, or simply because of disagreements among users and developers. Postponing decisions enables the
> project to move on smoothly and avoid confrontation with reality or peers. Unfortunately, difficult
> decisions will eventually need to be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation. Translating a system specification in to a formal or semi-formal model forces developers to identify and resolve difficult issues early in
> the development.

Question sources

Computer Science MCQ practice collection · Question 1

Question 166 | AM0002

Which design principle suggests that software components should have a single, well-defined responsibility?

A. DRY (Don't Repeat Yourself) principle

B. SOLID principle

C. KISS (Keep It Simple, Stupid) principle

D. YAGNI (You Ain't Gonna Need It) principle

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Single Responsibility, within SOLID.

SE25 includes Single Responsibility among the SOLID principles. That principle assigns a component one well-defined responsibility. DRY limits duplicated knowledge, KISS avoids unnecessary complexity and YAGNI avoids speculative work. Those can complement focused responsibilities, but none is the named principle directly described by the question.

Why each choice is right or wrong

A - Incorrect: DRY limits duplication of knowledge rather than specifically assigning one responsibility per component.

B - Correct: SOLID includes the Single Responsibility principle that the stem describes.

C - Incorrect: KISS favors simplicity, a different design concern from the specific single-responsibility rule.

D - Incorrect: YAGNI avoids speculative work rather than defining responsibility boundaries.

Primary note proof

Study-note section SE25

> Use abstraction and decomposition to manage complexity; preserve consistency, traceability and clear responsibilities. Top-down design refines high-level structure; bottom-up design assembles existing components. A hybrid combines both. SOLID includes Single Responsibility, Open-Closed, Liskov substitution, Interface Segregation and Dependency Inversion. Depend on abstractions; add compatible extensions; avoid forcing unused operations. DRY limits duplicated knowledge, KISS avoids unnecessary complexity, and YAGNI avoids speculative work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 120 | lines 24-27

> Coherence is the strength of dependencies within a subsystem. If a subsystem contains many objects that are related to each other and perform similar tasks, its coherence is high. If a subsystem
> contains a number of unrelated objects, its coherence is low. A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence.

Question sources

Computer Science MCQ practice collection · Question 2

Question 167 | AM0003

Which of the following is an agile development methodology?

A. Waterfall model

B. Scrum

C. V-model

D. Spiral model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Scrum.

SE05 presents Scrum through timeboxed Sprints, a Sprint Goal and a usable Increment, in the Agile context. Waterfall and the V-model organize planned stages and corresponding tests, while spiral is defined by risk-driven cycles. Therefore Scrum is the specifically Agile approach among the listed alternatives; iteration alone does not make every process Scrum.

Why each choice is right or wrong

A - Incorrect: Classical waterfall organizes planned phases rather than the Agile Scrum approach described.

B - Correct: Scrum is the Agile framework using Sprints and usable Increments.

C - Incorrect: The V-model pairs development stages with test levels rather than being the listed Agile framework.

D - Incorrect: Spiral is risk driven and is not the specifically Agile methodology named among these choices.

Primary note proof

Study-note section SE05

> Scrum uses timeboxed Sprints of at most one month, with a Sprint Goal, selected work and a usable Increment meeting a Definition of Done. Scope is discussed before and during planning; it is not forbidden to agree scope. Timeboxes tend to stay fixed while content is adapted without endangering the goal. Agile can be combined with required assurance/documentation; safety-critical context alone cannot prove it is unsuitable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules

to
follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Question sources

Computer Science MCQ practice collection · Question 3

Question 168 | AM0004

A team delivers small usable capabilities, replans the next work with customers and treats adaptation as a central value rather than organizing cycles around risk analysis. Which broad approach best fits?

A. Waterfall model

B. Agile methodology

C. Prototype model

D. Spiral model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Agile methodology

The team adds small usable capabilities and repeatedly replans with customers. Agile makes this combination of working results, customer participation and adaptation central. A prototype's immediate purpose is learning about requirements or feasibility; a spiral cycle is organized around risk reduction. Classical waterfall instead emphasizes planned phase commitments. Increments by themselves can occur in several processes, so the collaboration and planning priorities establish the intended comparison.

Why each choice is right or wrong

A - Incorrect: Classical waterfall does not make repeated adaptive replanning its organizing emphasis.

B - Correct: Agile combines working increments with ongoing customer collaboration and adaptation.

C - Incorrect: Prototyping primarily explores uncertainty rather than naming this ongoing delivery-and-planning organization.

D - Incorrect: Spiral's defining emphasis is risk analysis and reduction.

Primary note proof

Study-note section SE04

Iteration revisits and improves a solution; an increment adds usable functionality. A process may do both. Delivering registration first, then payments, then reporting is incremental; revising

a registration prototype after feedback is iterative. Each increment still needs design, implementation and testing.

Study-note section SE06

Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Study-note section SE05

Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Software Engineering module | PDF page 22 | lines 8-12

This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
also provide a platform for evaluation by the user.

Question sources

Computer Science MCQ practice collection · Question 4

Question 169 | AM0005

Which software testing technique focuses on finding defects by executing the software without knowledge of its internal structure?

A. White-box testing

B. Black-box testing

C. Integration testing

D. System testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: Black-box testing

Black-box equivalence testing chooses input classes based on expected behavior without tracing internal paths; white-box paths require source knowledge.

Why each choice is right or wrong

A - Incorrect: Requires code/data-structure knowledge for path selection. See PDF p. 149, 149.

B - Correct: Matches behavior testing without internal-structure knowledge. See PDF p. 149, 149.

C - Incorrect: A test level that can combine techniques. See PDF p. 149, 149.

D - Incorrect: A complete-system scope rather than the knowledge-based distinction. See PDF p. 149, 149.

Primary note proof

Study-note section SE36

> Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair. Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 149 | lines 6-10

> Equivalence testing is a black box testing technique that minimizes the number of test cases. The possible inputs are partitioned into equivalence classes, and a test case is selected for each class. The assumption of equivalence testing is that systems usually behave in similar ways for all members of a class. To test the behavior associated with an equivalence class, we only need to test
> one member of the class. Equivalence testing consists of two steps: identification of the

Software Engineering module | PDF page 149 | lines 24-29

Path testing
Path testing is a white box testing technique that identifies faults in the implementation of the component. The assumption behind path testing is that, by exercising all possible paths through the code at least once, most faults will trigger failures. The identification of paths requires knowledge of the source code and data structures.
The starting point for path testing is the flow graph. A flow graph consists of nodes representing

Question sources

Computer Science MCQ practice collection · Question 5

Question 170 | AM0006

Which diagrams describe detailed behavioral evolution over time through either ordered participant interactions or event-triggered state changes? Select all that apply.

A. Use case diagrams

B. Class diagrams

C. Sequence diagrams

D. State machine diagrams

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C, D: Sequence diagrams; State machine diagrams

Sequence diagrams show ordered messages across participant lifelines, while state machines show event-triggered transitions through an object's lifecycle states. Both therefore describe behavioral evolution over time under the stated scope. A use-case diagram summarizes external roles and goals without that detailed temporal evolution, and a class diagram describes static types and relationships. Neither replaces the two detailed dynamic views requested.

Why each choice is right or wrong

A - Incorrect: Use-case diagrams summarize roles/goals rather than detailed temporal evolution.

B - Incorrect: Class diagrams are static.

C - Correct: Sequence diagrams show ordered participant interactions.

D - Correct: State machines show event-triggered lifecycle changes.

Primary note proof

Study-note section SE20

Choose a sequence diagram for message order between participants, a state machine for event-dependent state changes, and activity for workflow. Both sequence and state diagrams describe behavior over time. State machines are useful for objects with significant states,

whether entity or control. A DFD shows data stores/flows and transformations from inputs to outputs; it is not a UML diagram.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 54 | lines 26-35

both buttons simultaneously. Figure 2.10 depicts a sequence diagram for an actor setting his 2Bwatch one minute ahead.
Each column represents an object that participates in the interaction. The vertical axis represents
time from top to bottom. Messages are shown by arrows. Labels on arrows represent message names and may contain arguments. Activations (i.e., executing methods) are depicted by vertical rectangles. Actors are shown as the leftmost column.
Sequence diagrams can be used to describe either an abstract sequence (i.e., all possible interactions) or concrete sequences (i.e., one possible interaction, as in Figure 2.10). When describing all possible interactions, sequence diagrams also provide notations for conditionals and
iterators.

Software Engineering module | PDF page 55 | lines 4-9

2.3.3. State chart Diagram
A UML statechart is a notation for describing the sequence of states an object goes through in response to external events. Statecharts are extensions of the finite state machine model. On the one hand, statecharts provide notation for nesting states and state machines (i.e., a state can be described by a state machine). On the other hand, statecharts provide notation for binding transitions with message sends and conditions on objects.

Question sources

Computer Science MCQ practice collection · Question 6

Question 171 | AM0007

Which software development methodology promotes adaptive planning, evolutionary development, and continuous improvement?

A. Waterfall model

B. Agile methodology

C. V-model

D. Spiral model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Agile methodology

Adaptive planning, evolutionary improvement and responsiveness to customer feedback characterize Agile as a broad development approach. Waterfall emphasizes planned phase progression; the V-model pairs development stages with corresponding test levels. Spiral can evolve a product as well, but its defining organization is repeated risk assessment and reduction. The question asks for the approach whose core emphasis matches the stated planning and collaboration values.

Why each choice is right or wrong

A - Incorrect: Waterfall emphasizes planned phase progression rather than adaptive planning.

B - Correct: Agile places adaptive planning and ongoing improvement at the center.

C - Incorrect: The V-model emphasizes correspondence between development and test stages.

D - Incorrect: Spiral is organized primarily around risk reduction, even when development is evolutionary.

Primary note proof

Study-note section SE05

> Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to

follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Software Engineering module | PDF page 22 | lines 8-12

This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
also provide a platform for evaluation by the user.

Question sources

Computer Science MCQ practice collection · Question 7

Question 172 | AM0008

Which technique is used to represent the flow of activities and decisions in a software system?

A. Use case modeling

B. Activity diagrams

C. State machine diagrams

D. Component diagrams

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: Activity diagrams

Activity diagrams interpret actions and transitions as control flow; decision diamonds branch that flow based on conditions.

Why each choice is right or wrong

A - Incorrect: Captures external functions, not detailed action flow. See PDF p. 56, 57.

B - Correct: Explicit action/control-flow notation with decisions. See PDF p. 56, 57.

C - Incorrect: Shows states responding to events rather than primarily the activity sequence. See PDF p. 56, 57.

D - Incorrect: Represents component organization rather than activity decisions. See PDF p. 56, 57.

Primary note proof

Study-note section SE20

An activity diagram models workflow: filled circle start, rounded actions, guarded decision diamond, thick fork/join bars and partitions/swimlanes for responsibility. A diamond can also

merge alternative flows; a join synchronizes concurrent flows. A state machine models states and transitions triggered by events, often with guards/actions, for one object's lifecycle.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 56 | lines 18-24

2.3.4. Activity diagrams
The outgoing transitions are triggered by the completion of an action associated with the state. This is called an action state. By convention, the name of a state denotes a condition, whereas the name of an action state denotes an action. Activity diagrams are statechart diagrams whose states
are action states. is an activity diagram corresponding to the state diagram. An alternate and equivalent view of activity diagrams is to interpret action states as control flow between activities
and transitions; that is, the arrows are interpreted as sequential constraints between activities.

Software Engineering module | PDF page 57 | lines 7-11

Decisions are branches in the control flow. They denote alternative transitions based on a condition
of the state of an object or a set of objects. Decisions are depicted by a diamond with one or more incoming arrows and two or more outgoing arrows. The outgoing arrows are labeled with the conditions that select a branch in the control flow. The set of all outgoing transitions from a decision represents the set of all possible outcomes.

Question sources

Computer Science MCQ practice collection · Question 8

Question 173 | AM0009

Developers build a quick interface version solely to discover how users perform an unfamiliar task; it is not a production release. Which named approach describes the immediate purpose of that version?

A. Waterfall model

B. Agile methodology

C. Prototype model

D. Spiral model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Prototype model

The early version is built to learn how users understand and carry out a task, not to deliver a production increment. That exploratory purpose identifies a prototype. Agile can use prototypes but its broader definition centers on collaboration and adaptation. Spiral may prototype to reduce a risk, but risk cycles are not the specific technique requested. Classical waterfall does not itself name early experimental versions. The purpose of the artifact distinguishes these overlapping practices.

Why each choice is right or wrong

A - Incorrect: Waterfall is a phased lifecycle rather than the named exploratory artifact technique.

B - Incorrect: Agile may use prototypes but does not specifically name the described usability experiment.

C - Correct: Prototyping uses an experimental version to discover requirements or usability needs.

D - Incorrect: Spiral may employ a prototype, but its defining method is organizing cycles around risk reduction.

Primary note proof

Study-note section SE07

> A prototype explores requirements, usability or technical feasibility. A throwaway prototype is discarded after learning; an evolutionary prototype is deliberately refined toward production. A quick prototype may omit robustness, security or architecture, so do not deploy it unchanged without evaluation. Coding may occur during prototyping even if a diagram labels that activity construction rather than coding.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 82 | lines 23-29

> • Prototype test. During this type of test, one or more users are presented with a piece of software that practically implements the system. A vertical prototype implements a complete use case through the system, and a horizontal prototype presents an interface for most use cases (without providing much or any functionality). The advantages of prototype tests are that they provide a realistic view of the system to the user and that prototypes can be instrumented to collect detailed data. The disadvantages of prototypes are that they are expensive to build and to modify.

Software Engineering module | PDF page 26 | lines 6-9

2. Requirement’s elicitation and analysis: This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on. This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

Question sources

Computer Science MCQ practice collection · Question 9

Question 174 | AM0010

Which testing technique focuses on testing the interaction between different components or modules of a software system?

A. Unit testing

B. Integration testing

C. System testing

D. Regression testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: Integration testing

Integration testing targets interfaces and interactions among components that have been combined. Unit testing isolates a smaller function or component, while system testing evaluates the complete integrated product against its requirements. Regression testing is triggered by a change and looks for unintended effects; it can be performed at several levels. The scenario is categorized by its interaction scope, so integration is the specific level requested.

Why each choice is right or wrong

A - Incorrect: Unit testing isolates a function or small component.

B - Correct: Integration testing exercises interactions among combined components.

C - Incorrect: System testing evaluates the complete product rather than specifically the component interfaces.

D - Incorrect: Regression names the after-change purpose, not a particular component-combination level.

Primary note proof

Study-note section SE35

Unit testing isolates a function, class or small component, often replacing dependencies with stubs/mocks; a driver invokes the unit and a stub substitutes a called dependency. Integration testing exercises interfaces and interactions among combined parts. System testing checks the

complete integrated system against functional and nonfunctional requirements in a representative environment.

Study-note section SE36

Functional tests check required behavior against the SRS and acceptance criteria. Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits. Usability, security, compatibility and reliability tests target their respective qualities. “Normal operating conditions” alone does not distinguish functional from performance testing.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Software Engineering module | PDF page 147 | lines 14-18

• Unit testing, which finds faults by isolating an individual component using test stubs and drivers and by exercising the component using a test case
• Integration testing, which finds faults by integrating several components together
• System testing, which focuses on the complete system, its functional and nonfunctional requirements, and its target environment

Question sources

Computer Science MCQ practice collection · Question 10

Question 175 | AM0011

Which design principle suggests that software entities should depend on abstractions, not on concrete implementations?

A. DRY (Don't Repeat Yourself) principle

B. SOLID principle

C. KISS (Keep It Simple, Stupid) principle

D. YAGNI (You Ain't Gonna Need It) principle

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Dependency Inversion, within SOLID.

SE25 lists Dependency Inversion within SOLID and explicitly directs designs to depend on abstractions. That reduces dependence on concrete implementation details and matches the stem. DRY, KISS and YAGNI address duplication, simplicity and speculative functionality respectively; none is the principle defining the abstraction-based dependency relationship.

Why each choice is right or wrong

A - Incorrect: DRY addresses duplicated knowledge rather than the direction and target of dependencies.

B - Correct: SOLID includes Dependency Inversion, which directs dependencies toward abstractions.

C - Incorrect: KISS addresses unnecessary complexity rather than specifying abstraction-based dependencies.

D - Incorrect: YAGNI limits speculative work rather than defining dependency direction.

Primary note proof

Study-note section SE25

> Use abstraction and decomposition to manage complexity; preserve consistency, traceability and clear responsibilities. Top-down design refines high-level structure; bottom-up design assembles existing components. A hybrid combines both. SOLID includes Single Responsibility, Open-Closed, Liskov substitution, Interface Segregation and Dependency Inversion. Depend on abstractions; add compatible extensions; avoid forcing unused operations. DRY limits duplicated knowledge, KISS avoids unnecessary complexity, and YAGNI avoids speculative work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 119 | lines 13-25

> 6.1.2. Services and subsystem interfaces
> A subsystem is characterized by the services it provides to other subsystems. A service is a set of

related operations that share a common purpose. A subsystem providing a notification service, for
example, defines operations to send notices, look up notification channels, and subscribe and unsubscribe to a channel.
The set of operations of a subsystem that are available to other subsystems form the subsystem interface. The subsystem interface, also referred to as the application programmer interface (API),
includes the name of the operations, their parameters, their types, and their return values. System
design focuses on defining the services provided by each subsystem that is, enumerating the operations, their parameters, and their high-level behavior. Object design will focus on defining the subsystem interfaces, that is, the type of the parameters and the return value of each operation.
The definition of a subsystem in terms of the services it provides helps us focus on its interface as opposed to its implementation. A good subsystem interface should provide as little information

Software Engineering module | PDF page 120 | lines 6-12

6.1.3. Coupling and coherence
Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Question sources

Computer Science MCQ practice collection · Question 11

Question 176 | AM0012

Which software development approach follows a linear, sequential flow from requirements gathering to testing and maintenance?

A. Waterfall model

B. Agile methodology

C. Prototype model

D. Spiral model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Waterfall model

Classical waterfall organizes development as planned sequential phases such as requirements, design, implementation and testing. Agile instead adapts work through feedback, prototyping uses experimental versions to learn, and spiral cycles organize work around risk reduction. Real waterfall projects can revisit work under change control, but the idealized linear organization described by the question is its characteristic model rather than one of the feedback-centered alternatives.

Why each choice is right or wrong

A - Correct: The idealized waterfall model follows the stated planned phase sequence.

B - Incorrect: Agile emphasizes adaptation and incremental planning.

C - Incorrect: Prototyping learns through experimental versions and feedback.

D - Incorrect: Spiral organizes repeated cycles around risk assessment and reduction.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 15 | lines 16-20

> non-iterative (or sequential) process in which progress is seen as flowing steadily downwards (like
> a waterfall) through the development phases. Thus the waterfall model maintains that one should
> move to a phase only when its preceding phase is reviewed and verified. That is, each phase must
> be completed before the next phase can begin. In waterfall model phases do not overlap. But there
> are various modified waterfall models, and hence, may include slight or major modifications.

Question sources

Computer Science MCQ practice collection · Question 12

Question 177 | AM0013

Which technique is used to estimate the size and complexity of a software project?

A. Use case modeling

B. Storyboarding

C. Function point analysis

D. Pair programming

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer C: Function point analysis

Function-point analysis counts system externals, assigns complexity weights and aggregates them to estimate software size independent of implementation language.

Why each choice is right or wrong

A - Incorrect: Captures functionality but is not the stated weighted size-estimation method. See PDF p. 93, 91.

B - Incorrect: A user-experience representation, not the module's quantitative size method. See PDF p. 93, 91.

C - Correct: Directly taught size/complexity weighting technique. See PDF p. 93, 91.

D - Incorrect: A development collaboration practice, not this sizing method. See PDF p. 93, 91.

Primary note proof

Study-note section SE28

> Function points estimate functional size using inputs, outputs, inquiries, internal logical files and external interface files, weighted under the chosen method. They are not just lines of code and are not a direct exact measure of complexity. Git tracks versions; issue trackers such as Jira track work/defects; CI tools such as Jenkins automate checks; Docker packages application/runtime dependencies into containers.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 93 | lines 3-21

> • Function points allow the measurement of software size in standard units, independent of the underlying language in which the software is developed.
> • Counting the number of externals (inputs, outputs, inquiries, and interfaces) that make up the system.
> • There are five types of externals to count:
> 1. External inputs - data or control inputs (input files, tables, forms, screens, messages,

etc.) to the system
2. External outputs - data or control outputs from the system
3. External inquiries - I/O queries which require a response (prompts, interrupts, calls, etc.)
4. External interfaces - libraries or programs which are passed into and out of the system (I/O routines, sorting procedures, math libraries, run-time libraries, etc.)
5. Internal data files - groupings of data stored internally in the system (entities, internal control files, directories)
Apply these steps to calculate the size of a project:
1. Count or estimate all the occurrences of each type of external.
2. Assign each occurrence a complexity weight.
3. Multiply each occurrence by its complexity weight, and total the results to obtain a function count. Complexity weights are as listed below.

Software Engineering module | PDF page 91 | lines 37-42

• Software Metrics can be used to indicate the basic attributes of the software. The indicative metrics, not exhaustive, are
• Size - Line of code ,Function point count
• Quality- Reliability, Accuracy, Dependability, and these are measured through some measures like errors, failures and number of runs between failures ,etc.
• Execution Time -process time of execution of a critical process and many more.

Question sources

Computer Science MCQ practice collection · Question 13

Question 178 | AM0014

Which tool is used to track and manage software issues, bugs, and tasks throughout the development process?

A. Git

B. Jira

C. Jenkins

D. Docker

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer B: Jira.

SE28 distinguishes version tracking, issue tracking, automated checks and container packaging. It names Jira as an issue tracker for work and defects, matching bugs and tasks across development. Git records versions, Jenkins automates checks and Docker packages runtime dependencies; those tools may integrate with issue tracking but have different primary roles.

Why each choice is right or wrong

A - Incorrect: Git tracks source versions and history rather than serving as the named issue/task tracker.

B - Correct: Jira tracks work items, defects and tasks through their development workflow.

C - Incorrect: Jenkins automates checks and integration work rather than primarily managing issue records.

D - Incorrect: Docker packages runtime dependencies into containers rather than tracking bugs and tasks.

Primary note proof

Study-note section SE28

> Function points estimate functional size using inputs, outputs, inquiries, internal logical files and external interface files, weighted under the chosen method. They are not just lines of code and are not a direct exact measure of complexity. Git tracks versions; issue trackers such as Jira track work/defects; CI tools such as Jenkins automate checks; Docker packages application/runtime dependencies into containers.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 5-10

> Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
> faults if change is controlled.

Software Engineering module | PDF page 90 | lines 7-24

> 4.2. Project Planning
> • Probably the most time-consuming project management activity.
> • Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
> • Various different types of plans may be developed to support the main software project plan that is concerned with schedule and budget.
> The project plan sets out:
> • The resources available to the project;
> • The work breakdown;
> • A schedule for the work.
> Project Plan Structure
> • Introduction.
> • Project organisation.
> • Risk analysis.
> • Hardware and software resource requirements.

- Work breakdown.
- Project schedule.
- Monitoring and reporting mechanisms.

Question sources

Computer Science MCQ practice collection · Question 14

Question 179 | AM0015

Which design principle suggests that software entities should be open for extension but closed for modification?

A. DRY (Don't Repeat Yourself) principle

B. SOLID principle

C. KISS (Keep It Simple, Stupid) principle

D. YAGNI (You Ain't Gonna Need It) principle

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Open-Closed, within SOLID.

SE25 lists Open-Closed within SOLID and recommends adding compatible extensions. The stem quotes that principle's central idea: support extension while keeping established code stable. DRY, KISS and YAGNI address other concerns. Open-Closed is a design goal achieved through suitable abstractions, rather than a claim that software can never need modification.

Why each choice is right or wrong

A - Incorrect: DRY limits duplicated knowledge rather than defining openness for extension.

B - Correct: SOLID includes the Open-Closed principle explicitly described by the stem.

C - Incorrect: KISS favors simple solutions rather than the specific extension-versus-modification relationship.

D - Incorrect: YAGNI avoids speculative functionality rather than defining the Open-Closed design goal.

Primary note proof

Study-note section SE25

Use abstraction and decomposition to manage complexity; preserve consistency, traceability and clear responsibilities. Top-down design refines high-level structure; bottom-up design assembles existing components. A hybrid combines both. SOLID includes Single Responsibility, Open-Closed, Liskov substitution, Interface Segregation and Dependency Inversion. Depend on

abstractions; add compatible extensions; avoid forcing unused operations. DRY limits duplicated knowledge, KISS avoids unnecessary complexity, and YAGNI avoids speculative work.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 104 | lines 6-17

5.1.4. Generalization
Generalization enables us to organize concepts in to hierarchies. There may be any number of intermediate levels in between, covering more-or-less generalized concepts (e.g., Low Priority Incident, Emergency, Disaster). Such hierarchies allow us to refer to many concepts precisely. When
we use the term Incident, we mean all instances of all types of Incidents. When we use the term Emergency, we only refer to an Incident that requires an immediate response. This view of generalization stems from modeling.
In an object-oriented programming language, inheritance is a reusability technique. If a class Child
inherits from a class Parent, all the attributes and methods available on Parent are automatically available on Child. The class Child may add additional methods or override inherited methods, thus refining the class Parent. As in the case of generalization, the class at the top of the hierarchy
tends to be the most general one, whereas the leaves tend to be the most specialized.

Software Engineering module | PDF page 32 | lines 12-21

A class in an object-oriented system carefully delineates between its interfaces the routines and attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact
on other classes, and so, the impact is minimized. However, the object-oriented approach is not a panacea; nothing is magical here that will promote perfect design or perfect code.
Promotion of reusability
Objects are reusable because they are modeled directly out of a real-world problem domain. Each
object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.

Question sources

Computer Science MCQ practice collection · Question 15

Question 180 | AM0016

Which technique is used to represent the static structure of a software system, including classes, relationships, and attributes?

A. Use case diagrams

B. Class diagrams

C. Sequence diagrams

D. State machine diagrams

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: Class diagrams

Class diagrams explicitly depict classes with attributes and operations and their associations/generalizations.

Why each choice is right or wrong

A - Incorrect: External functions and actors. See PDF p. 51.

B - Correct: Exact structural modeling notation. See PDF p. 51.

C - Incorrect: Time-ordered messages among objects. See PDF p. 51.

D - Incorrect: State evolution rather than static attributes/relationships. See PDF p. 51.

Primary note proof

Study-note section SE17

> A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links. UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process. Structural things describe conceptual or physical structural elements.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 51 | lines 14-24

> 2.3.2. Class Diagrams
> Classes and objects
> Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behavior of a set of objects. Objects are entities that encapsulate state and behavior. Each object has an identity: It can be referred individually and is distinguishable from other objects.
> In UML, classes and objects are depicted by boxes including three compartments. The top compartment displays the name of the class or object. The center compartment displays its

attributes, and the bottom compartment displays its operations. The attribute and operation compartment can be omitted for clarity. Object names are underlined to indicate that they are instances. By convention, class names start with an uppercase letter. Objects in object diagrams

Question sources

Computer Science MCQ practice collection · Question 16

Question 181 | AM0017

Customer representatives stay in close contact with developers and use feedback to reshape the next timeboxed work. Which methodology makes this collaboration and adaptation a foundational priority?

A. Waterfall model

B. Agile methodology

C. V-model

D. Spiral model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Agile methodology

Frequent stakeholder communication alone is possible in several lifecycles. The scenario additionally gives customer representatives ongoing access to the team and lets feedback reshape the next timeboxed body of work, matching Agile's collaboration and adaptation values. Waterfall and the V-model primarily organize planned stages, while spiral organizes commitments around risk reduction. The combination of priorities, rather than communication frequency by itself, identifies the intended methodology.

Why each choice is right or wrong

A - Incorrect: Waterfall is defined by planned phase progression, not the stated adaptive collaboration priority.

B - Correct: Agile explicitly prioritizes customer collaboration and responding to change.

C - Incorrect: The V-model is defined by linked development and testing stages.

D - Incorrect: Spiral's organizing priority is risk assessment and reduction, though stakeholders can participate.

Primary note proof

Study-note section SE05

Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter

still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Study-note section SE03

Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE06

Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Software Engineering module | PDF page 22 | lines 8-12

This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
also provide a platform for evaluation by the user.

Software Engineering module | PDF page 81 | lines 11-21

3.4.2. Negotiating specifications with clients: Joint Application Design
Joint Application Design (JAD) is effectiveness lies in that the requirements elicitation work is done in one single workshop session in which all stakeholders participate. Users, clients, developers, and a trained session leader sit together in one room to present their viewpoint, listen
to other viewpoints, negotiate, and agree on a mutually acceptable solution. The outcome of the workshop, the final JAD document, is a complete system specification document that includes definitions of data elements, work flows, and interface screens. Because the final document is jointly developed by the stakeholders (that is, the participants who not only have an interest in the

success of the project, but also can make substantial decisions) the final JAD document represents
an agreement between users, clients, and developers, and thus minimizes requirements changes later in the development process.

Question sources

Computer Science MCQ practice collection · Question 17

Question 182 | AM0018

Which technique documents an actor’s goal with preconditions, a main success flow, alternative flows and postconditions?

A. Use case modeling

B. Storyboarding

C. Wireframing

D. Pair programming

Hint

Match the information to be represented with each model’s purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer A: Use case modeling

A use-case description captures an actor’s goal together with preconditions, main flow, alternatives and postconditions. This provides the structured behavioral requirements record in the scenario. Storyboards illustrate moments in an interaction and wireframes sketch interface structure; either can help discovery without supplying that full behavioral contract. Pair programming organizes two people’s implementation work. It is therefore use-case modeling that directly names the requested requirements representation.

Why each choice is right or wrong

A - Correct: Use-case modeling records actors, goals and structured successful and alternative flows.

B - Incorrect: Storyboarding sketches interaction moments rather than necessarily supplying the full goal-and-flow description.

C - Incorrect: Wireframing sketches interface structure and layout.

D - Incorrect: Pair programming is collaborative implementation work.

Primary note proof

Study-note section SE14

An actor is an external role interacting with the system; it can be a person, device or another system. A person may play several roles, and several people may play one role. A primary actor

seeks a goal; a supporting actor provides a service. Ask which user groups perform main functions, administer or maintain the system, and which external systems exchange information.

Study-note section SE-S05

A storyboard sketches a sequence of moments in a proposed interaction, helping people explore how a concept would be used. A wireframe sketches interface structure and the arrangement of content or controls. Both can inform requirements, but neither automatically supplies a complete use-case description with actors, preconditions, flows and postconditions.

Study-note section SE-S05

Pair programming is a way for two programmers to work together on the same implementation task, commonly sharing a workstation with complementary writing and reviewing roles. It is an implementation collaboration practice rather than a replacement for modeling a user goal.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
Use cases describe the behavior of the system as seen from an actor's point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
use case to access the system functionality. The use case can then initiate other use cases and gather
more information from the actors. When actors and use cases exchange information, they are said
to communicate. We will see later that we represent these exchanges with communication relationships.

Software Engineering module | PDF page 48 | lines 18-32

To describe a use case, we use a template composed of six fields (see also Figure 2.2):
• The name of the use case is unique across the system so that developers (and project participants) can unambiguously refer to the use case.
• Participating actors are actors interacting with the use case.
• Entry conditions describe the conditions that need to be satisfied before the use case is initiated.
• The flow of events describes the sequence of actions of the use case, which are be numbered for reference. The common case (i.e., cases that occur frequently) and the exceptional cases (i.e., cases that seldom occur, such as errors and unusual conditions) are described separately in different use cases for clarity.
• Exit conditions describe the conditions that are satisfied after the completion of the use case.
• Special requirements are requirements that are not related to the functionality of the

system. These include constraints on the performance of the system, its implementation, the hardware platforms it runs on, and so on.

Question sources

Computer Science MCQ practice collection · Question 18

Question 183 | AM0019

Which testing technique focuses on retesting modified or affected parts of the software to ensure that changes do not introduce new defects?

A. Unit testing

B. Integration testing

C. Regression testing

D. System testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: Regression testing.

SE36 defines regression testing by the purpose of detecting unintended effects after change. That includes retesting changed and affected areas to find newly introduced faults. Unit, integration and system testing describe levels or scope; regression tests can be run at any of them, so the change-driven purpose uniquely identifies C.

Why each choice is right or wrong

A - Incorrect: Unit testing defines a small-component scope rather than the purpose of checking change-induced defects.

B - Incorrect: Integration testing focuses on interfaces and interactions, which may be regression-tested but do not define regression.

C - Correct: Regression testing retests previously working behavior to detect unintended effects of modifications.

D - Incorrect: System testing addresses the complete integrated system rather than specifically the after-change purpose.

Primary note proof

Study-note section SE36

Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair.

Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 147 | lines 14-18

• Unit testing, which finds faults by isolating an individual component using test stubs and drivers and by exercising the component using a test case
• Integration testing, which finds faults by integrating several components together
• System testing, which focuses on the complete system, its functional and nonfunctional requirements, and its target environment

Software Engineering module | PDF page 142 | lines 5-10

Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
faults if change is controlled.

Software Engineering module | PDF page 147 | lines 3-8

• A test stub is a partial implementation of components on which the tested component depends. A test driver is a partial implementation of a component that depends on the tested component. Test stubs and drivers enable components to be isolated from the rest of the system for testing.
• A correction is a change to a component. The purpose of a correction is to repair a fault. Note that a correction can introduce new faults.

Question sources

Computer Science MCQ practice collection · Question 19

Question 184 | AM0020

Which tool is used to package an application and its dependencies into a container for easy deployment and scalability?

A. Git

B. Jira

C. Jenkins

D. Docker

Hint

Distinguish the planning or organization goal from the tool or structure used to support it. Apply the stated project conditions.

Answer D: Docker.

SE28 explicitly says Docker packages an application and its runtime dependencies into containers. That matches the deployment packaging task in the question. Git tracks versions, Jira tracks issues and Jenkins automates checks; they can participate in a deployment pipeline, but none replaces Docker's specified container packaging responsibility.

Why each choice is right or wrong

A - Incorrect: Git manages version history rather than packaging the runtime into a container.

B - Incorrect: Jira manages work and defects rather than building application containers.

C - Incorrect: Jenkins automates checks and pipelines but is not the named container-packaging tool.

D - Correct: Docker packages the application and runtime dependencies into a deployable container.

Primary note proof

Study-note section SE28

> Function points estimate functional size using inputs, outputs, inquiries, internal logical files and external interface files, weighted under the chosen method. They are not just lines of code and are not a direct exact measure of complexity. Git tracks versions; issue trackers such as Jira track work/defects; CI tools such as Jenkins automate checks; Docker packages application/runtime dependencies into containers.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 124 | lines 8-16

> Client/server architecture
> In the client/server architecture (Figure 6.9), a subsystem, the server, provides servicesto instances
> of other subsystems called the clients, which are responsible for interacting with theuser. The request for a service is usually done via a remote procedure call mechanism or acommon object broker (e.g., CORBA or Java RMI). Control flow in the clients and the servers isindependent except for synchronization to manage requests or to receive results.
> Figure 6.9 Client/server architecture (UML class diagram).
> Clients request services from one or more servers. The server has no knowledge of the client. The client/server architecture is a generalization of the repository architecture.

Software Engineering module | PDF page 119 | lines 13-25

> 6.1.2. Services and subsystem interfaces
> A subsystem is characterized by the services it provides to other subsystems. A service is a set of

related operations that share a common purpose. A subsystem providing a notification service, for
example, defines operations to send notices, look up notification channels, and subscribe and unsubscribe to a channel.
The set of operations of a subsystem that are available to other subsystems form the subsystem interface. The subsystem interface, also referred to as the application programmer interface (API),
includes the name of the operations, their parameters, their types, and their return values. System
design focuses on defining the services provided by each subsystem that is, enumerating the operations, their parameters, and their high-level behavior. Object design will focus on defining the subsystem interfaces, that is, the type of the parameters and the return value of each operation.
The definition of a subsystem in terms of the services it provides helps us focus on its interface as opposed to its implementation. A good subsystem interface should provide as little information

Question sources

Computer Science MCQ practice collection · Question 20

Question 185 | AM0021

Which phase of the software development life cycle (SDLC) focuses on gathering and understanding user requirements?

A. Analysis

B. Design

C. Implementation

D. Testing

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer A: Analysis

Elicitation and analysis establish user needs before internal design and implementation.

Why each choice is right or wrong

A - Correct: Corresponds to gathering and structuring requirements. See PDF p. 62, 100.

B - Incorrect: Plans the internal realization. See PDF p. 62, 100.

C - Incorrect: Realizes the software design in code. See PDF p. 62, 100.

D - Incorrect: Tries to expose faults in implemented software. See PDF p. 62, 100.

Primary note proof

Study-note section SE08

Business requirements explain why the project exists and its goals. User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements. A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 62 | lines 9-16

Requirement's elicitation focuses on describing the purpose of the system. The client, the developers, and the users identify a problem area and define a system that addresses the problem.
Such a definition is called a system specification and serves as a contract between the client and the developers. The system specification is structured and formalized during analysis to produce an analysis model (see Figure 3.1). Both system specification and analysis model represent the same information. They differ only in the language and notation they use. The system specification
is written in natural language, whereas the analysis model is usually expressed in a formal or semiformal notation.

Software Engineering module | PDF page 100 | lines 11-24

Analysis focuses on producing a model of the system, called the analysis model, which is correct, complete, consistent, and verifiable. Analysis is different from requirements elicitation in that developers focus on structuring and formalizing the requirements elicited from users
Figure 4.1 Products of requirements elicitation and analysis (UML activity diagram).
Althoughtheanalysismodelmaynotbeunderstandabletotheusersandtheclient, it helps developers verify the system specification produced during requirements elicitation. There is a natural tendency for users and developers to postpone difficult decisions until later in the project. A decision may be difficult because of lack of domain knowledge, lack of technological knowledge, or simply because of disagreements among users and developers. Postponing decisions enables the
project to move on smoothly and avoid confrontation with reality or peers. Unfortunately, difficult
decisions will eventually need to be made, often at higher cost when intrinsic problems are discovered during testing, or worse, during user evaluation. Translating a system specification in to a formal or semi-formal model forces developers to identify and resolve difficult issues early in
the development.

Question sources

Computer Science MCQ practice collection · Question 21

Question 186 | AM0022

What is the primary goal of software design?

A. To create a visually appealing user interface

B. To optimize the performance of the software system

C. To translate user requirements into a detailed plan

D. To develop and test individual software components

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: To translate user requirements into a detailed plan

Design transforms the SRS into a structure suitable for implementation and identifies internal architecture and responsibilities.

Why each choice is right or wrong

A - Incorrect: UI appearance is one design concern, not the whole goal. See PDF p. 18, 116.

B - Incorrect: Performance optimization is a design goal, not the full transformation purpose. See PDF p. 18, 116.

C - Correct: Matches transforming requirements into an implementable design. See PDF p. 18, 116.

D - Incorrect: Coding/testing occur after or alongside design refinement, not its primary deliverable. See PDF p. 18, 116.

Primary note proof

Study-note section SE21

> System design turns the analysis model into architecture: design goals, subsystem responsibilities, dependencies, hardware mapping, data management, access control, control flow and startup/shutdown/error policies. Detail/object design refines APIs, parameter/return types, exceptions, component choices and algorithms. A subsystem is a manageable logical collection with services and an interface; it need not be a physical replaceable artifact.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 18 | lines 13-20

> System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
> Software design involves identifying and describing the fundamental software system abstractions
> and their relationships. The goal of the design phase is to transform the requirements specified in
> the SRS document into a structure that is suitable for implementation in some programming

language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Software Engineering module | PDF page 116 | lines 16-29

The analysis model describes the system completely from the actors' point of view and serves as the basis of communication between the client and the developers. The analysis model, however, does not contain information about the internal structure of the system, its hardware configuration,
or, more generally, how the system should be realized. System design is the first step in this direction. System design results in the following products:
• List of design goals, describing the qualities of the system that developers should optimize
• Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.
The design goals are derived from the nonfunctional requirements. Design goals guide the decisions to be made by the developers especially when trade-offs are needed. The subsystem decomposition constitutes the bulk of system design. Developers divide the system into manageable pieces to deal with complexity: Each subsystem is assigned to a team and realized independently. In order for this to be possible, though, developers need to address system-wide

Question sources

Computer Science MCQ practice collection · Question 22

Question 187 | AM0023

Which broad approach explicitly centers its values on collaboration, working results and responding to change, using frequent feedback to improve later work?

A. Waterfall model

B. Agile methodology

C. Spiral model

D. V-model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Agile methodology

The stated priorities combine short feedback cycles, collaboration and adaptation as central development values. Agile names that broad approach. Waterfall and the V-model emphasize planned stages, with the V-model additionally relating stages to test levels. Spiral can also iterate and consult stakeholders, but its defining cycle is risk-driven. Distinguishing the core emphasis avoids incorrectly claiming that only Agile can communicate or revisit work.

Why each choice is right or wrong

A - Incorrect: Classical waterfall emphasizes planned phase commitments.

B - Correct: Agile makes collaboration and responsiveness to change central values.

C - Incorrect: Spiral is defined by explicit risk-reduction cycles rather than this value-based emphasis.

D - Incorrect: The V-model is defined by development-test correspondences.

Primary note proof

Study-note section SE05

> Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to

follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Software Engineering module | PDF page 22 | lines 8-12

This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
also provide a platform for evaluation by the user.

Question sources

Computer Science MCQ practice collection · Question 23

Question 188 | AM0024

A team separates a reusable reporting component behind a stable interface so other applications can use it and its implementation can change without rewriting clients. Which benefit most directly captures both goals?

A. To improve the readability of the code

B. To ensure efficient memory management

C. To facilitate code reuse and maintenance

D. To minimize the number of software defects

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: To facilitate code reuse and maintenance

A module with a stable interface can be reused in another application, and hiding its implementation allows localized changes without rewriting its clients. Those are reuse and maintenance benefits. Readability may improve as well, but it does not fully describe the two goals stated in the scenario. Modularization does not automatically optimize memory use or eliminate defects; good boundaries and verification still matter.

Why each choice is right or wrong

A - Incorrect: Readability may improve, but it does not fully describe reuse across applications and localized change.

B - Incorrect: Memory efficiency is not guaranteed by separating modules.

C - Correct: Reuse and maintenance directly match the two stated goals.

D - Incorrect: Reducing defects can be a benefit, but it does not name the stated reuse-and-change objective.

Primary note proof

Study-note section SE22

Logical cohesion groups similar-category operations, often selected by a flag; temporal cohesion groups operations done at the same time; functional cohesion focuses on one task. Splitting an unrelated three-purpose component improves modularity. If behavior is preserved, the restructuring is refactoring; the words are not mutually exclusive answers. Smaller modules can aid understanding, independent work, reuse and maintenance, but too many interfaces also create complexity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 118 | lines 11-16

6.1.1. Subsystems and classes
In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Software Engineering module | PDF page 32 | lines 12-21

A class in an object-oriented system carefully delineates between its interfaces the routines and attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact
on other classes, and so, the impact is minimized. However, the object-oriented approach is not a panacea; nothing is magical here that will promote perfect design or perfect code.
Promotion of reusability
Objects are reusable because they are modeled directly out of a real-world problem domain. Each
object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.

Software Engineering module | PDF page 120 | lines 6-12

6.1.3. Coupling and coherence
Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Question sources

Computer Science MCQ practice collection · Question 24

Question 189 | AM0025

Which type of testing aims to uncover defects in the interaction between different components of a software system?

A. Unit testing

B. Integration testing

C. System testing

D. Regression testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: Integration testing

The defects sought are in communication or interfaces among combined components, so the applicable level is integration testing. Unit tests concentrate on isolated units and system tests assess the complete product. Regression testing instead describes checking for unintended effects after a change, and can include unit, integration or system tests. The distinction is between the scope of combination and the purpose that triggers a test run.

Why each choice is right or wrong

A - Incorrect: Unit testing concentrates on an isolated function or small component.

B - Correct: Integration testing targets interactions among combined parts.

C - Incorrect: System testing checks the whole integrated system against requirements.

D - Incorrect: Regression is an after-change objective that can occur at several testing levels.

Primary note proof

Study-note section SE35

> Unit testing isolates a function, class or small component, often replacing dependencies with stubs/mocks; a driver invokes the unit and a stub substitutes a called dependency. Integration testing exercises interfaces and interactions among combined parts. System testing checks the complete integrated system against functional and nonfunctional requirements in a representative environment.

Study-note section SE36

> Functional tests check required behavior against the SRS and acceptance criteria. Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits. Usability, security, compatibility and reliability tests

target their respective qualities. “Normal operating conditions” alone does not distinguish functional from performance testing.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Question sources

Computer Science MCQ practice collection · Question 25

Question 190 | AM0026

Which programming paradigm is based on the concept of objects and their interactions?

A. Functional programming

B. Procedural programming

C. Object-oriented programming

D. Event-driven programming

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Object-oriented programming

Object-oriented programming organizes a system around objects that combine state and behavior and interact through operations. Procedural programming decomposes work into functions or procedures that manipulate data. Functional programming emphasizes function evaluation and controlled side effects, while event-driven programming organizes reactions to events. An object-oriented program may also use events or functions, but the question asks which paradigm is defined by objects and their interactions.

Why each choice is right or wrong

A - Incorrect: Functional programming emphasizes evaluation of functions and controlled side effects.

B - Incorrect: Procedural programming emphasizes functions or procedures operating on data.

C - Correct: Object-oriented programming centers on objects with state and behavior.

D - Incorrect: Event-driven programming centers on reactions to events, independent of object-based decomposition.

Primary note proof

Study-note section SE01

> Procedural decomposition organizes functions; object oriented decomposition organizes objects with state and behavior. Either can use iterative or sequential development. OOP does not automatically shorten every project. Reuse, encapsulation and consistent models can reduce duplication and ease change; poor object design can still be complex. Functional programming emphasizes function evaluation and controlled side effects; event-driven programs react to events.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 40 | lines 12-18

> ✓ Object -Oriented paradigm is an approach to the solution of problems in which all computations are performed in the context of objects.
> ✓ Object-oriented programming allows decomposition of a problem into a number of objects and then builds data and functions around these objects.
> ✓ In an object-oriented environment everything is an object and each object is responsible for itself.
> ✓ Object oriented systems are easier to adapt to changes, more robust, and easier to maintain.

Question sources

Computer Science MCQ practice collection · Question 26

Question 191 | AM0027

What is the defining artifact-management function of version control, distinguishing it from collaboration benefits and CI services?

A. To manage project schedules and milestones

B. To facilitate collaboration among team members

C. To track and control changes to source code and files

D. To automate the testing and deployment processes

Hint

Separate artifact history and controlled changes from scheduling, collaboration benefits and automated execution.

Answer C: To track and control changes to source code and files

Version control records artifact versions and their history, making it possible to inspect changes, maintain branches and combine compatible work. Collaboration is an important benefit, but it is broader than the defining version-history function. Scheduling belongs to project planning, while automated testing and deployment belong to CI or delivery tooling that can use version-control events. These roles can be integrated without becoming the same activity.

Why each choice is right or wrong

A - Incorrect: Scheduling and milestones belong to project planning and tracking.

B - Incorrect: Collaboration is a benefit enabled by shared versions and merging, rather than the defining artifact-history function.

C - Correct: Tracking and controlling file changes is the direct version-control function.

D - Incorrect: Testing and deployment automation is CI/delivery work that may integrate with version control.

Primary note proof

Study-note section SE33

> Configuration management identifies controlled items, versions/baselines, change procedures, status and audits. Items may include source code, requirements, manuals, test plans, environments and build configurations. A team’s ordinary browser is usually a tool rather than product CI, but a controlled test-browser version may be an environment CI.

Study-note section SE28

> Git tracks versions; issue trackers such as Jira track work/defects; CI tools such as Jenkins automate checks; Docker packages application/runtime dependencies into containers.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 142 | lines 5-10

Configuration management avoids faults caused by undisciplined change in the system models. For example, it is a common mistake to a change a subsystem interface without notifying all developers of calling components. Configuration management can make sure that, if analysis models and code are becoming inconsistent with one another, analysts and implementor are notified. The assumption behind configuration management is that the system contains many less
faults if change is controlled.

Question sources

Computer Science MCQ practice collection · Question 27

Question 192 | AM0028

Which conventional lifecycle is described by a simple linear requirements→design→implementation→testing sequence with reviewed phase completion, without the V-model's explicit development-to-test pairing?

A. Waterfall model

B. Agile methodology

C. Spiral model

D. V-model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Waterfall model

Classical waterfall organizes work as separated planned phases with reviewed deliverables, matching the stated linear sequence. The V-model additionally pairs development artifacts with later test levels, which this question excludes. Agile emphasizes adaptation and feedback, while spiral organizes repeated risk-driven cycles. Explicitly distinguishing the test-pairing structure removes the overlap between two staged processes without denying their relationship.

Why each choice is right or wrong

A - Correct: Classical waterfall matches the specified simple linear organization.

B - Incorrect: Agile emphasizes adaptive content and feedback.

C - Incorrect: Spiral uses risk-driven cycles.

D - Incorrect: The V-model adds explicit development-to-test pairing excluded here.

Primary note proof

Study-note section SE03

The V-model pairs specification/design stages with later test levels. Requirements inform acceptance tests; system design informs system tests; component/detail design informs

integration/unit tests. Coding lies at the bottom. Test planning starts early. Stable requirements and strong assurance may favor this organization, but neither a V shape nor waterfall guarantees safety or correctness.

Study-note section SE03

Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 15 | lines 16-20

non-iterative (or sequential) process in which progress is seen as flowing steadily downwards (like
a waterfall) through the development phases. Thus the waterfall model maintains that one should
move to a phase only when its preceding phase is reviewed and verified. That is, each phase must
be completed before the next phase can begin. In waterfall model phases do not overlap. But there
are various modified waterfall models, and hence, may include slight or major modifications.

Question sources

Computer Science MCQ practice collection · Question 28

Question 193 | AM0029

What is the primary purpose of software documentation?

A. To serve as a record of code authorship

B. To provide instructions for end-users

C. To assist in software maintenance and understanding

D. To ensure compliance with coding standards

Hint

Distinguish process assurance from product evaluation, and compare specified obligations with actual stakeholder needs.

Answer C: To assist in software maintenance and understanding

Documentation records requirements and architecture so developers can understand decisions, revisit designs and coordinate later change. A user manual is one audience-specific part of this broader purpose.

Why each choice is right or wrong

A - Incorrect: Authorship alone does not capture system decisions or behavior. See PDF p. 130, 83, 32.

B - Incorrect: End-user instructions are a valid subset, not the overall technical purpose. See PDF p. 130, 83, 32.

C - Correct: Matches understanding, revisiting and maintaining documented structures. See PDF p. 130, 83, 32.

D - Incorrect: Coding-standard compliance is narrower than the general understanding purpose. See PDF p. 130, 83, 32.

Primary note proof

Study-note section SE34

Quality assurance establishes and checks processes/standards; quality control evaluates work products. An SQA plan identifies standards, responsibilities, evaluations, reviews, audits and assurance records. Quality of conformance means following the design/specification; a product can conform to a poor specification. Verification asks whether specified obligations are implemented correctly; validation asks whether the product meets intended stakeholder needs. Both can use reviews, analysis or execution, so “human-only versus computer-only” is false.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 130 | lines 3-11

6.3. DOCUMENTING SYSTEM DESIGN
System design is documented in the System Design Document (SDD). It describes design goals set by the project, the subsystem decomposition (with UML class diagrams), the hardware/software mapping (with UML deployment diagrams), the data management, the access control, control flow mechanisms, and boundary conditions. The SDD is used to define interfaces between teams of developers and as reference when architecture-level decisions need to be revisited. The audience for the SDD includes the project management, the system architects (i.e., the developers who participate in the system design), and the developers who design and implement each subsystem. The following is an example template for a SDD:

Software Engineering module | PDF page 83 | lines 12-25

3.4.4. Documenting Requirements Elicitation
The results of the requirements elicitation activity and the analysis activity are documented in the
Requirements Analysis Document (RAD). This document completely describes the system in terms of functional and nonfunctional requirements and serves as a contractual basis between the
client and the developers. The audience for the RAD includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design). The first part of the document, including use cases and nonfunctional requirements, is written during requirements

elicitation. The formalization of the specification in terms of object models is written during analysis.
The first section of the RAD is an Introduction. Its purpose is to provide a brief overview of the function of the system and the reasons for its development, its scope, and references to the development context (e.g., related problem statement, references to existing systems, feasibility studies). The introduction also includes the objectives and success criteria of the project.

Software Engineering module | PDF page 32 | lines 12-21

A class in an object-oriented system carefully delineates between its interfaces the routines and attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact
on other classes, and so, the impact is minimized. However, the object-oriented approach is not a panacea; nothing is magical here that will promote perfect design or perfect code.
Promotion of reusability
Objects are reusable because they are modeled directly out of a real-world problem domain. Each
object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.

Question sources

Computer Science MCQ practice collection · Question 29

Question 194 | AM0030

After each short usable release, customers reorder the remaining work and the team adapts the next release’s content while keeping its timebox. Which broad approach most directly fits this planning behavior?

A. Waterfall model

B. Agile methodology

C. Spiral model

D. V-model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Agile methodology

A fixed short timebox with usable results and customer-driven adaptation of later content fits Agile planning. Classical waterfall and the V-model organize planned development stages, with the latter pairing stages to tests. Spiral cycles organize commitments around risk assessment and reduction. All can incorporate some staged work, so the stated treatment of feedback and later scope is what makes the Agile option the most direct match.

Why each choice is right or wrong

A - Incorrect: Classical waterfall emphasizes planned phases and later delivery feedback.

B - Correct: Agile adapts later work from customer feedback while producing usable increments.

C - Incorrect: Spiral is specifically organized around risk reduction, not the stated timeboxed adaptation rule.

D - Incorrect: The V-model is defined by paired development and test stages.

Primary note proof

Study-note section SE04

> Choose incremental delivery when prioritized partial capability must reach users early. Smaller changes can be easier to debug and permit feedback, though architecture and integration still require planning. Iteration need not rebuild the whole system every time, and increments need not be perfectly independent modules. Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Study-note section SE05

> Scrum uses timeboxed Sprints of at most one month, with a Sprint Goal, selected work and a usable Increment meeting a Definition of Done. Scope is discussed before and during planning; it is not forbidden to agree scope. Timeboxes tend to stay fixed while content is adapted without endangering the goal. Agile can be combined with required assurance/documentation; safety-critical context alone cannot prove it is unsuitable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 22 | lines 8-12

> This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
> also provide a platform for evaluation by the user.

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
> follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Question sources

Computer Science MCQ practice collection · Question 30

Question 195 | AM0031

Which design principle advocates for designing software components that have a single responsibility?

A. Encapsulation

B. Inheritance

C. Polymorphism

D. Single Responsibility Principle

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: Single Responsibility Principle.

SE25 identifies Single Responsibility among the core design principles. The question asks directly for one responsibility per component, so D is the exact named rule. Encapsulation protects internal decisions, inheritance creates subtype relations, and polymorphism supports varying implementations through a common interface. Those can help a design but do not themselves define a single responsibility.

Why each choice is right or wrong

A - Incorrect: Encapsulation hides implementation details rather than requiring one responsibility per component.

B - Incorrect: Inheritance establishes superclass/subclass reuse rather than the focused-responsibility rule.

C - Incorrect: Polymorphism supports compatible varying implementations rather than defining responsibility count.

D - Correct: Single Responsibility explicitly calls for one focused responsibility in a software component.

Primary note proof

Study-note section SE25

> SOLID includes Single Responsibility, Open-Closed, Liskov substitution, Interface Segregation and Dependency Inversion.

Study-note section SE-S02

> Generalization places common concepts in a more general superclass and specialized concepts in subclasses. Available inherited attributes and operations are shared by subclasses, which may extend or override behavior.

Study-note section SE-S02

> Encapsulation groups state with operations and hides representation behind an interface. Polymorphism permits different object types to respond differently through a common operation or interface. Abstraction exposes the relevant contract while omitting unnecessary detail. Reusability means using suitable existing classes or components in new contexts rather than writing them again. These ideas support each other but name different properties.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 120 | lines 24-27

> Coherence is the strength of dependencies within a subsystem. If a subsystem contains many objects that are related to each other and perform similar tasks, its coherence is high. If a subsystem
> contains a number of unrelated objects, its coherence is low. A desirable property of a subsystem decomposition is that it leads to subsystems with high coherence.

Software Engineering module | PDF page 101 | lines 9-15

> The analysis object model consists of entity, boundary, and control objects. Entity objects represent the persistent information tracked by the system. Boundary objects represent the interactions between the actors and the system. Control objects represent the tasks that a formed by the user and supported by the system. In the 2B watch example, Year, Month, Day are entity objects; Button Boundary and LCD Display Boundary are boundary objects; Change Date Control is
> a control object that represents the activity of changing the date by pressing combinations of buttons.

Question sources

Computer Science MCQ practice collection · Question 31

Question 196 | AM0032

Which design pattern is used to ensure that only one instance of a class is created throughout the system?

A. Singleton pattern

B. Factory pattern

C. Observer pattern

D. Decorator pattern

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer A: Singleton.

SE27 describes Singleton as controlling access to one instance. That is the pattern requested by the one-instance condition. Factory patterns address how concrete objects are created, Observer notifies subscribers and Decorator adds behavior by wrapping. None of those other pattern responsibilities guarantees a single shared instance by itself.

Why each choice is right or wrong

A - Correct: Singleton controls creation and access so that the intended class has one shared instance.

B - Incorrect: A factory selects or constructs products rather than inherently limiting a class to one instance.

C - Incorrect: Observer distributes notifications to subscribers rather than restricting object instance count.

D - Incorrect: Decorator adds behavior through wrapping rather than enforcing a single instance.

Primary note proof

Study-note section SE27

> Factory Method lets subclasses choose a concrete product through a creation operation, reducing coupling to concrete constructors. A simple factory centralizes creation but is not identical to Factory Method. Singleton controls access to one instance; Observer notifies subscribers; Memento stores/restores state while hiding representation; Iterator traverses; Decorator adds behavior through wrapping. The pattern name is Memento, not Momento.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 121 | lines 6-15

6.1.4. Software architecture
As the complexity of systems increases, the specification of the system decomposition is critical. It is difficult to modify or correct a weak decomposition once development has started, as most subsystem interfaces have to change. In recognition of the importance of this problem, the concept
of software architecture has emerged. A software architecture includes the system decomposition, the global control flow, error-handling policies and inter subsystem communication protocols.
In this section, we describe a few sample architectures that can be used for different systems. This
is by no means a systematic or thorough exposition of the subject. Rather, we aim to provide a few
representative examples and refer the reader to the literature for more details.

Question sources

Computer Science MCQ practice collection · Question 32

Question 197 | AM0033

Which test type checks that specified services produce the required outputs under normal operating conditions, without measuring speed/resources or user task ease?

A. Stress testing

B. Performance testing

C. Functional testing

D. Usability testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: Functional testing

Functional tests compare observable services and outputs with the SRS and acceptance criteria. That is the stated objective, so C applies. Performance testing measures speed, throughput or resources, and usability evaluates user task success or ease; neither is requested here. Stress testing exceeds normal limits. Normal operating conditions alone would not distinguish these types, but the explicit behavioral objective does.

Why each choice is right or wrong

A - Incorrect: Stress testing exceeds normal operating limits.

B - Incorrect: Performance testing measures speed, throughput or resources.

C - Correct: Functional testing checks specified services and outputs.

D - Incorrect: Usability testing evaluates user task ease or success.

Primary note proof

Study-note section SE36

Functional tests check required behavior against the SRS and acceptance criteria. Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits. Usability, security, compatibility and reliability tests target their respective qualities. “Normal operating conditions” alone does not distinguish functional from performance testing.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Software Engineering module | PDF page 79 | lines 3-9

• Performance characteristics. How responsive should the system be? How many concurrent users should it support? What is a typical or extreme load?
• Error handling and extreme conditions. How should the system handle exceptions? Which exceptions should the system handle? What is the worse environment in which the system is expected to perform? Are there safety requirements on the system?
• Quality issues. How reliable/available/robust should the system be? What is the client’s involvement in assessing the quality of the system or the development process?

Question sources

Computer Science MCQ practice collection · Question 33

Question 198 | AM0034

Which programming paradigm promotes the use of data and avoids side effects?

A. Functional programming

B. Procedural programming

C. Object-oriented programming

D. Event-driven programming

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Functional programming, for the side-effect emphasis.

The phrase promotes the use of data is not distinctive because all these paradigms use data. The discriminating part is avoiding or controlling side effects. SE01 explicitly assigns function evaluation and controlled side effects to functional programming. Procedural, object-oriented and event-driven styles organize behavior differently and do not inherently impose that emphasis.

Why each choice is right or wrong

A - Correct: Functional programming emphasizes evaluating functions and controlling side effects.

B - Incorrect: Procedural decomposition organizes functions or procedures without inherently avoiding state-changing effects.

C - Incorrect: Object-oriented decomposition combines state and behavior and does not inherently exclude side effects.

D - Incorrect: Event-driven programs react to events; that does not itself require avoiding side effects.

Primary note proof

Study-note section SE01

> Procedural decomposition organizes functions; object oriented decomposition organizes objects with state and behavior. Either can use iterative or sequential development. OOP does not automatically shorten every project. Reuse, encapsulation and consistent models can reduce duplication and ease change; poor object design can still be complex. Functional programming emphasizes function evaluation and controlled side effects; event-driven programs react to events.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 40 | lines 12-18

✓ Object -Oriented paradigm is an approach to the solution of problems in which all computations are performed in the context of objects.
✓ Object-oriented programming allows decomposition of a problem into a number of objects and then builds data and functions around these objects.
✓ In an object-oriented environment everything is an object and each object is responsible for itself.
✓ Object oriented systems are easier to adapt to changes, more robust, and easier to maintain.

Question sources

Computer Science MCQ practice collection · Question 34

Question 199 | AM0035

Which type of software maintenance involves making changes to a software system to adapt it to new environments or technologies?

A. Corrective maintenance

B. Adaptive maintenance

C. Perfective maintenance

D. Preventive maintenance

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer B: Adaptive maintenance.

SE40 defines adaptive maintenance as responding to changes in the environment. New platforms or technologies therefore create an adaptive-maintenance need even when the original software is not faulty. Corrective maintenance fixes faults, perfective work improves useful capability or performance, and preventive work reduces future risk. The stated environmental motive makes B the appropriate category.

Why each choice is right or wrong

A - Incorrect: Corrective work fixes faults rather than primarily accommodating changed environments or technology.

B - Correct: Adaptive maintenance changes software so it continues working in a new environment or technology context.

C - Incorrect: Perfective work improves capability or performance rather than specifically responding to an environmental change.

D - Incorrect: Preventive work reduces future maintenance risk rather than defining adaptation to a new environment.

Primary note proof

Study-note section SE40

> Corrective maintenance fixes faults; adaptive maintenance responds to environment changes; perfective maintenance improves useful capability/performance; preventive maintenance reduces future risk, such as restructuring fragile code. Motive determines category; refactoring can support preventive or perfective work. Maintenance often lasts years and includes understanding legacy code and regression effort, so it may dominate lifecycle cost.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 19 | lines 24-29

> 5. Operation and maintenance: Normally (although not necessarily), this is the longest life cycle phase. Maintenance of a typical software product requires much more than the effort necessary to
> develop the product itself. The system is installed and put into practical use. Maintenance involves
> correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system ‘s services as new requirements are discovered.

Software Engineering module | PDF page 41 | lines 16-17

> ✓ Software evolution takes place when you change existing software systems to meet new requirements. Changes are continuous and the software must evolve to remain useful.

Question sources

Computer Science MCQ practice collection · Question 35

Question 200 | AM0036

What is the purpose of usability testing in software development?

A. To evaluate the performance of the software system under stress conditions

B. To assess the user-friendliness and ease of use of the software system

C. To verify the compliance of the software system with specified requirements

D. To identify and fix defects or issues in the software system

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer B: To assess the user-friendliness and ease of use of the software system

Usability testing examines ease of learning, task completion and user errors, with the goal of identifying and improving usability problems.

Why each choice is right or wrong

A - Incorrect: Stress performance is not the ease-of-use focus. See PDF p. 82.

B - Correct: Directly matches ease of use and learning. See PDF p. 82.

C - Incorrect: General requirements conformance is broader than this specific user-focused test. See PDF p. 82.

D - Incorrect: General fault fixing is broader; the test identifies user interaction difficulties rather than implementing all repairs. See PDF p. 82.

Primary note proof

Study-note section SE36

> Functional tests check required behavior against the SRS and acceptance criteria. Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits. Usability, security, compatibility and reliability tests target their respective qualities. “Normal operating conditions” alone does not distinguish functional from performance testing.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 82 | lines 3-12

> though this type of approach could be used for any parameter, usability tests focus on usability parameters, such as the ease to learn the system, the time to accomplish a task, or the rate of errors
> a user makes when accomplishing a task.
> There are two important differences between classical experiments and usability tests. Whereas the classical experimental method is designed to confirm or refute a hypothesis, the goal of usability tests is to obtain qualitative information on how to fix usability problems and how to improve the system. Another difference is the rigor with which the experiments need to be performed. It has been shown that even a series of quick focused tests starting as early as requirements elicitation is extremely helpful. Nielsen uses the term “discount usability engineering” to indicate that a few usability tests are better than none at all.

Question sources

Computer Science MCQ practice collection · Question 36

Question 201 | AM0037

Which software development methodology involves iterative development cycles called sprints?

A. Waterfall model

B. Agile methodology

C. Spiral model

D. V-model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Agile methodology, specifically Scrum.

SE05 places Sprints in Scrum, an Agile framework. B is therefore the available category that contains the specific process described. Agile as a whole does not require every method to call its iterations Sprints. Waterfall, spiral and the V-model have other organizing principles, so none is the named Sprint-based method.

Why each choice is right or wrong

A - Incorrect: Classical waterfall uses planned phases rather than Scrum Sprints.

B - Correct: Scrum is an Agile framework whose iterative delivery cycles are called Sprints.

C - Incorrect: Spiral uses risk-driven cycles rather than specifically the Scrum Sprint mechanism.

D - Incorrect: The V-model pairs development stages with tests rather than organizing work into Sprints.

Primary note proof

Study-note section SE05

> Scrum uses timeboxed Sprints of at most one month, with a Sprint Goal, selected work and a usable Increment meeting a Definition of Done. Scope is discussed before and during planning; it is not forbidden to agree scope. Timeboxes tend to stay fixed while content is adapted without endangering the goal. Agile can be combined with required assurance/documentation; safety-critical context alone cannot prove it is unsuitable.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

> Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
> follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Question sources

Computer Science MCQ practice collection · Question 37

Question 202 | AM0038

What is the primary purpose of code refactoring?

A. To add new features to the software system

B. To optimize the performance of the software system

C. To improve the readability and maintainability of the code

D. To fix defects or issues in the software system

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: To improve the readability and maintainability of the code

Restructuring explicitly addresses maintainability, readability and understandability, while optimization separately addresses speed and memory. C matches refactoring's purpose of improving internal structure and maintainability.

Why each choice is right or wrong

A - Incorrect: Adds behavior, whereas restructuring reorganizes existing design elements. See PDF p. 133.

B - Incorrect: Optimization is separately classified by performance goals. See PDF p. 133.

C - Correct: Direct match to the documented restructuring goals. See PDF p. 133.

D - Incorrect: Repairing a fault is correction; restructuring need not fix a behavioral defect. See PDF p. 133.

Primary note proof

Study-note section SE43

> Pareto's 80/20 rule is a prioritization heuristic: a small subset of causes may account for much of the observed impact. It does not guarantee 80% of defects can be fixed with exactly 20% effort. Refactor to preserve behavior while improving structure; optimize only against measured needs; record rationale and update traceability and regression evidence after change.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 133 | lines 3-15

> transformation on subsets of a particular model. Typical activities include transforming n-ary associations into binary associations, implementing binary associations as references, merging two similar classes from two different subsystems into a single class, collapsing classes with no significant behavior into attributes, splitting complex classes into simpler ones, rearranging classes and operations to increase the inheritance and packaging. During restructuring, we address design goals such as maintainability, readability, and understandability of the system model.

• Optimization. Optimization activities address performance requirements of the system model. This includes changing algorithms to respond to speed or memory requirements, reducing multiplicities in associations to speed up queries, adding redundant associations for efficiency, rearranging execution orders, adding derived attributes to improve the access time to objects and opening up the architecture, that is, adding access to lower layers because of performance requirements.

Question sources

Computer Science MCQ practice collection · Question 38

Question 203 | AM0039

Which approach treats customer participation in reviewing working results and reprioritizing future work as a foundational value, rather than merely a practice a project may choose?

A. Waterfall model

B. Agile methodology

C. Spiral model

D. V-model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Agile methodology

The project treats customer participation in reviewing results and reprioritizing future work as a foundational development value. This directly matches Agile's customer collaboration and responsiveness to change. Waterfall and the V-model primarily define phase and test-stage organization. Spiral also includes stakeholder review, but risk assessment and reduction define its cycle. Participation alone is not exclusive; the explicit priority given to continual adaptive collaboration identifies the answer.

Why each choice is right or wrong

A - Incorrect: Classical waterfall's defining organization is planned phase progression.

B - Correct: Agile explicitly prioritizes customer collaboration and adapting to feedback.

C - Incorrect: Spiral includes review but is defined by risk-driven cycles.

D - Incorrect: The V-model organizes linked development and test stages, rather than naming the stated collaboration values.

Primary note proof

Study-note section SE05

Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Study-note section SE03

Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE06

Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 13 | lines 22-27

Sometimes, software processes are categorized as either plan-driven or agile processes. Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan. In agile processes planning is incremental and it is easier to change the process to reflect changing customer requirements. There is no hard and fast rules to
follow but each approach is suitable for different types of software. Generally, you need to find a balance between plan-driven and agile processes.

Software Engineering module | PDF page 22 | lines 8-12

This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
also provide a platform for evaluation by the user.

Software Engineering module | PDF page 81 | lines 11-21

3.4.2. Negotiating specifications with clients: Joint Application Design
Joint Application Design (JAD) is effectiveness lies in that the requirements elicitation work is done in one single workshop session in which all stakeholders participate. Users, clients, developers, and a trained session leader sit together in one room to present their viewpoint, listen
to other viewpoints, negotiate, and agree on a mutually acceptable solution. The outcome of the workshop, the final JAD document, is a complete system specification document that includes definitions of data elements, work flows, and interface screens. Because the final document is

jointly developed by the stakeholders (that is, the participants who not only have an interest in the
success of the project, but also can make substantial decisions) the final JAD document represents
an agreement between users, clients, and developers, and thus minimizes requirements changes later in the development process.

Question sources

Computer Science MCQ practice collection · Question 39

Question 204 | AM0040

Which type of testing involves running the entire software system in an environment that closely resembles the production environment?

A. Unit testing

B. Integration testing

C. System testing

D. Regression testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: System testing

System testing evaluates the complete integrated software against its functional and nonfunctional requirements in a representative environment. Unit testing isolates a smaller element and integration testing concentrates on interfaces among combined parts. Regression testing instead checks for unintended consequences after change and is not restricted to any one level or environment. The question specifies the complete-product scope, making system testing the direct classification.

Why each choice is right or wrong

A - Incorrect: Unit testing isolates a function, class or small component.

B - Incorrect: Integration testing focuses on interfaces and interactions among combined parts.

C - Correct: System testing evaluates the complete integrated product in a representative environment.

D - Incorrect: Regression names an after-change checking purpose and can run at multiple levels.

Primary note proof

Study-note section SE35

Unit testing isolates a function, class or small component, often replacing dependencies with stubs/mocks; a driver invokes the unit and a stub substitutes a called dependency. Integration testing exercises interfaces and interactions among combined parts. System testing checks the complete integrated system against functional and nonfunctional requirements in a representative environment.

Study-note section SE36

Functional tests check required behavior against the SRS and acceptance criteria. Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits. Usability, security, compatibility and reliability tests target their respective qualities. “Normal operating conditions” alone does not distinguish functional from performance testing.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 147 | lines 14-18

• Unit testing, which finds faults by isolating an individual component using test stubs and drivers and by exercising the component using a test case
• Integration testing, which finds faults by integrating several components together
• System testing, which focuses on the complete system, its functional and nonfunctional requirements, and its target environment

Software Engineering module | PDF page 144 | lines 17-32

Figure 6.2 depicts an overview of the testing activities:
• Unit testing tries to find faults in participating objects and/or subsystems with respect to the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Question sources

Computer Science MCQ practice collection · Question 40

Question 205 | AM0041

What is a use case in use case modeling?

A. A description of a specific user interface design

B. A step-by-step procedure for software installation

C. A sequence of activities performed by a user or system

D. A diagram that represents the relationships between software components

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: A sequence of activities performed by a user or system

A use case describes a system-provided function as a flow of events producing an actor-visible result. C is acceptable when understood as goal-related interactions, not any arbitrary sequence of activities.

Why each choice is right or wrong

A - Incorrect: UI layout is an implementation/interface design artifact. See PDF p. 47, 48.

B - Incorrect: Installation instructions are one possible documented procedure, not the general use-case definition. See PDF p. 47, 48.

C - Correct: Best matches the goal-oriented flow of events involving actors and the system. See PDF p. 47, 48.

D - Incorrect: Component relationships describe internal structure rather than actor-visible behavior. See PDF p. 47, 48.

Primary note proof

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

> Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
> Use cases describe the behavior of the system as seen from an actor's point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate

a
use case to access the system functionality. The use case can then initiate other use cases and gather
more information from the actors. When actors and use cases exchange information, they are said
to communicate. We will see later that we represent these exchanges with communication relationships.

Software Engineering module | PDF page 48 | lines 18-32

To describe a use case, we use a template composed of six fields (see also Figure 2.2):
• The name of the use case is unique across the system so that developers (and project participants) can unambiguously refer to the use case.
• Participating actors are actors interacting with the use case.
• Entry conditions describe the conditions that need to be satisfied before the use case is initiated.
• The flow of events describes the sequence of actions of the use case, which are be numbered for reference. The common case (i.e., cases that occur frequently) and the exceptional cases (i.e., cases that seldom occur, such as errors and unusual conditions) are described separately in different use cases for clarity.
• Exit conditions describe the conditions that are satisfied after the completion of the use case.
• Special requirements are requirements that are not related to the functionality of the system. These include constraints on the performance of the system, its implementation, the hardware platforms it runs on, and so on.

Question sources

Computer Science MCQ practice collection · Question 41

Question 206 | AM0042

What is an actor in use-case modeling when the system boundary is fixed?

A. A component that performs specific actions in a software system

B. An external interacting role played by a person, device or another system.

C. A diagram that represents the flow of activities in a use case

D. A particular individual user account rather than a role.

Hint

Match the information to be represented with each model’s purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: An external interacting role played by a person, device or another system.

An actor is an external role that interacts across the chosen system boundary. A person, device or another system may play that role. It is not an internal component, a diagram, or merely one individual user account. Because actors model roles rather than particular people, one person can play multiple actors and multiple people can fulfill the same actor role.

Why each choice is right or wrong

A - Incorrect: An internal component is not an external actor merely because it acts.

B - Correct: The external interacting role is the actor.

C - Incorrect: An activity-flow diagram is a model artifact rather than an actor.

D - Incorrect: An individual account is not the role abstraction.

Primary note proof

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Study-note section SE14

> An actor is an external role interacting with the system; it can be a person, device or another system. A person may play several roles, and several people may play one role.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

> Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
> Use cases describe the behavior of the system as seen from an actor’s point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
> use case to access the system functionality. The use case can then initiate other use cases and gather
> more information from the actors. When actors and use cases exchange information, they are said
> to communicate. We will see later that we represent these exchanges with communication relationships.

Question sources

Computer Science MCQ practice collection · Question 42

Question 207 | AM0043

What is the purpose of an include relationship between use cases?

A. To represent the reuse of common steps between use cases

B. To represent optional steps within a use case

C. To represent the extension of a use case with additional steps

D. To represent the relationship between actors and use cases

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer A: To represent the reuse of common steps between use cases

Factoring common functionality into one included use case eliminates redundant descriptions, as in shared HelpDispatcher behavior.

Why each choice is right or wrong

A - Correct: Direct reuse/commonality mechanism. See PDF p. 50, 50.

B - Incorrect: Optional exceptional behavior is more directly represented by extend. See PDF p. 50, 50.

C - Incorrect: Adds events under conditions through extend rather than reuse via include. See PDF p. 50, 50.

D - Incorrect: Actor/use-case communication uses associations, not include. See PDF p. 50, 50.

Primary note proof

Study-note section SE16

> Include factors required reused behavior: Submit Order includes Validate Order, with the dashed arrow from the including/base use case to the included one. Extend adds conditional behavior at extension points: Apply Discount extends Checkout, with the arrow from the extension to the base. The base is meaningful independently. For an extension, draw the arrow from the extending use case to the base use case.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 50 | lines 3-8

> can be modeled by a use case HelpDispatcher that is included by the use cases OpenIncident and AllocateResources (and any other use cases accessible by the Dispatcher). The resulting model only describes the HelpDispatcher functionality once, thus reducing complexity. Two use cases are related by an include relationship if one of them includes the second one in its flow of events. In UML, include relationships are depicted by a dashed arrow originating from the including use case. Include relationships are labeled with the string <<include>>.

Software Engineering module | PDF page 50 | lines 14-24

Extend relationships
Extend relationships are an alternate means for reducing complexity in the use case model. A use
case can extend another use case by adding events. An extend relationship indicates that an instance of an extended use case may include (under certain conditions) the behavior specified by
the extending use case. A typical application of extend relationships is the specification of exceptional behavior. For example (Figure 2-19), assume that the network connection between the
Dispatcher and the FieldOfficer can be interrupted at any time. (e.g., if the FieldOfficer enters a tunnel). The use case ConnectionDown describes the set of events taken by the system and the actors while the connection is lost. ConnectionDown extends the use cases OpenIncident and AllocateResources. Separating exceptional behavior from common behavior enables us to write shorter and more focused use cases.

Question sources

Computer Science MCQ practice collection · Question 43

Question 208 | AM0044

Which statements describe extend between use cases? Select all that apply. Read optional as conditional behavior at an extension point.

A. To factor required reused steps into an included use case.

B. To represent optional steps within a use case

C. To represent the extension of a use case with additional steps

D. To represent the relationship between actors and use cases

Hint

Match the information to be represented with each model’s purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B, C: To represent optional steps within a use case; To represent the extension of a use case with additional steps

Extend adds conditional behavior at extension points in an independently meaningful base use case. It therefore represents optional additional behavior and extends the base with added steps, making both B and C true. Factoring required reused behavior is the include relationship. Associations between actors and use cases are separate from the extend relationship linking use cases to one another.

Why each choice is right or wrong

A - Incorrect: Required reused behavior is factored with include.

B - Correct: Conditional additional behavior need not occur in every base execution.

C - Correct: Extend adds steps to an independently meaningful base.

D - Incorrect: Actor association is a different relationship.

Primary note proof

Study-note section SE16

> Include factors required reused behavior: Submit Order includes Validate Order, with the dashed arrow from the including/base use case to the included one. Extend adds conditional behavior at extension points: Apply Discount extends Checkout, with the arrow from the extension to the base. The base is meaningful independently. For an extension, draw the arrow from the extending use case to the base use case.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 50 | lines 14-24

> Extend relationships
> Extend relationships are an alternate means for reducing complexity in the use case model. A use
> case can extend another use case by adding events. An extend relationship indicates that an instance of an extended use case may include (under certain conditions) the behavior specified by
> the extending use case. A typical application of extend relationships is the specification of exceptional behavior. For example (Figure 2-19), assume that the network connection between the
> Dispatcher and the FieldOfficer can be interrupted at any time. (e.g., if the FieldOfficer enters a tunnel). The use case ConnectionDown describes the set of events taken by the system and the actors while the connection is lost. ConnectionDown extends the use cases OpenIncident and AllocateResources. Separating exceptional behavior from common behavior enables us to write shorter and more focused use cases.

Software Engineering module | PDF page 50 | lines 3-8

> can be modeled by a use case HelpDispatcher that is included by the use cases OpenIncident and AllocateResources (and any other use cases accessible by the Dispatcher). The resulting model only describes the HelpDispatcher functionality once, thus reducing complexity. Two use cases are related by an include relationship if one of them includes the second one in its flow of events. In UML, include relationships are depicted by a dashed arrow originating from the including use case. Include relationships are labeled with the string <<include>>.

Question sources

Computer Science MCQ practice collection · Question 44

Question 209 | AM0045

Which diagram is commonly used to visualize the relationships between actors and use cases in use case modeling?

A. Class diagram

B. Use case diagram

C. Activity diagram

D. Sequence diagram

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: Use case diagram

Use-case diagrams depict actors and use cases with communication associations; these are the requested external interaction relationships.

Why each choice is right or wrong

A - Incorrect: Describes class structure. See PDF p. 47, 48.

B - Correct: Direct actor/use-case representation. See PDF p. 47, 48.

C - Incorrect: Describes action/control flow. See PDF p. 47, 48.

D - Incorrect: Describes time-ordered object interactions, not the high-level actor/function map. See PDF p. 47, 48.

Primary note proof

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

> Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
> Use cases describe the behavior of the system as seen from an actor's point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
> use case to access the system functionality. The use case can then initiate other use cases and

gather
more information from the actors. When actors and use cases exchange information, they are said
to communicate. We will see later that we represent these exchanges with communication relationships.

Software Engineering module | PDF page 48 | lines 18-32

To describe a use case, we use a template composed of six fields (see also Figure 2.2):
• The name of the use case is unique across the system so that developers (and project participants) can unambiguously refer to the use case.
• Participating actors are actors interacting with the use case.
• Entry conditions describe the conditions that need to be satisfied before the use case is initiated.
• The flow of events describes the sequence of actions of the use case, which are be numbered for reference. The common case (i.e., cases that occur frequently) and the exceptional cases (i.e., cases that seldom occur, such as errors and unusual conditions) are described separately in different use cases for clarity.
• Exit conditions describe the conditions that are satisfied after the completion of the use case.
• Special requirements are requirements that are not related to the functionality of the system. These include constraints on the performance of the system, its implementation, the hardware platforms it runs on, and so on.

Question sources

Computer Science MCQ practice collection · Question 45

Question 210 | AM0046

What defines an abstract use case, rather than optionality, unfinished documentation or reuse alone?

A. To represent a use case that is not yet fully defined

B. To represent a use case that is optional in the system

C. It specifies behavior for specialization and is not directly instantiated as a complete use case.

D. To represent a use case that is reused by other use cases

Hint

Match the information to be represented with each model’s purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: It specifies behavior for specialization and is not directly instantiated as a complete use case.

An abstract use case specifies behavior for specialization and is not directly instantiated as a complete use case. That is different from being unfinished, optional or reused. Concrete behavior can be included for reuse, and extend can express conditional additions. Nor does the absence of a direct actor line by itself establish abstraction. Option C states the actual defining criterion.

Why each choice is right or wrong

A - Incorrect: An incomplete description is not the abstraction criterion.

B - Incorrect: Optional behavior is not the abstraction criterion.

C - Correct: Specialization without direct complete instantiation defines this abstraction.

D - Incorrect: Reuse alone can also involve concrete use cases.

Primary note proof

Study-note section SE16

> Generalization relates a specialized actor/use case to a more general one, with a hollow triangle toward the general element. An abstract use case specifies behavior for specialization and is not directly instantiated as a complete use case. It is not defined merely by reuse or lack of a direct actor line. Included behavior is incorporated where the inclusion is reached; do not infer every conditional branch in the base always executes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 51 | lines 4-13

> Generalization relationships
> Generalization/specialization relationships are a third mechanism for reducing the complexity of a model. A use case can specialize a more general one by adding more detail. For example, FieldOfficers are required to authenticate before they can use FRIEND. During early stages of requirements elicitation, authentication is modeled as a high-level Authenticate use case. Later, developers describe the Authenticate use case in more detail and allow for several different hardware platforms. This refinement activity results in two more use cases, AuthenticateWithPasswordwhich enables FieldOfficers to login without any specific hardware, and AuthenticateWithCard, which enables FieldOfficers to login using a smart card. The two new use cases are represented as specializations of the Authenticate use case.

Software Engineering module | PDF page 50 | lines 3-8

> can be modeled by a use case HelpDispatcher that is included by the use cases OpenIncident and AllocateResources (and any other use cases accessible by the Dispatcher). The resulting model only describes the HelpDispatcher functionality once, thus reducing complexity. Two use cases are related by an include relationship if one of them includes the second one in its flow of events. In UML, include relationships are depicted by a dashed arrow originating from the including use case. Include relationships are labeled with the string <<include>>.

Question sources

Computer Science MCQ practice collection · Question 46

Question 211 | AM0047

Which option states the most specific defining purpose of an exception flow, distinguishing it from a generic alternative flow?

A. To represent an error or exceptional situation in the use case

B. To represent an alternative path within the use case

C. To represent the termination of the use case

D. To represent the relationship between primary and secondary actors

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer A: To represent an error or exceptional situation in the use case

An exception flow handles an error or failure encountered in the use case. Such a path is also an alternative to the main success flow in a broad sense, but B alone does not distinguish exceptional handling from normal conditional variations. An exception may allow correction and recovery instead of terminating the use case. Actor relationships do not define its failure-handling purpose.

Why each choice is right or wrong

A - Correct: Error or exceptional handling is the specific defining purpose.

B - Incorrect: Alternative path is broader and does not identify the exceptional nature.

C - Incorrect: Exceptions need not terminate the entire use case.

D - Incorrect: Actor relationships do not describe failure handling.

Primary note proof

Study-note section SE15

> Example: Submit Incident. Precondition: officer authenticated. 1 Officer supplies location and description. 2 System validates details. 3 System records the incident. 4 System returns its ID. Alternative 2a: location missing → report the problem and request correction; save nothing. Success postcondition: incident and ID persist; failure guarantee: no partial record. An alternate flow is a conditional variation; an exception flow handles a failure. Preconditions hold before entry, while a trigger starts the use case.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 48 | lines 18-32

To describe a use case, we use a template composed of six fields (see also Figure 2.2):
• The name of the use case is unique across the system so that developers (and project participants) can unambiguously refer to the use case.
• Participating actors are actors interacting with the use case.
• Entry conditions describe the conditions that need to be satisfied before the use case is initiated.
• The flow of events describes the sequence of actions of the use case, which are be numbered for reference. The common case (i.e., cases that occur frequently) and the exceptional cases (i.e., cases that seldom occur, such as errors and unusual conditions) are described separately in different use cases for clarity.
• Exit conditions describe the conditions that are satisfied after the completion of the use case.
• Special requirements are requirements that are not related to the functionality of the system. These include constraints on the performance of the system, its implementation, the hardware platforms it runs on, and so on.

Software Engineering module | PDF page 50 | lines 14-24

Extend relationships
Extend relationships are an alternate means for reducing complexity in the use case model. A use
case can extend another use case by adding events. An extend relationship indicates that an instance of an extended use case may include (under certain conditions) the behavior specified by
the extending use case. A typical application of extend relationships is the specification of exceptional behavior. For example (Figure 2-19), assume that the network connection between the
Dispatcher and the FieldOfficer can be interrupted at any time. (e.g., if the FieldOfficer enters a tunnel). The use case ConnectionDown describes the set of events taken by the system and the actors while the connection is lost. ConnectionDown extends the use cases OpenIncident and AllocateResources. Separating exceptional behavior from common behavior enables us to write shorter and more focused use cases.

Question sources

Computer Science MCQ practice collection · Question 47

Question 212 | AM0048

What is the primary goal of use case modeling?

A. To define the software architecture of the system

B. To identify and document user requirements

C. To design the user interface of the system

D. To implement the functionality of the system

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: To identify and document user requirements

Use cases capture externally visible functionality and let developers validate requirements with users, rather than defining internal architecture or implementation.

Why each choice is right or wrong

A - Incorrect: System design defines architecture after the external analysis model. See PDF p. 47, 62.

B - Correct: Directly captures actor-visible functional requirements. See PDF p. 47, 62.

C - Incorrect: UI layout can be explored alongside use cases but is not their primary purpose. See PDF p. 47, 62.

D - Incorrect: Implementation realizes the requirements instead of documenting them. See PDF p. 47, 62.

Primary note proof

Study-note section SE14

> An actor is an external role interacting with the system; it can be a person, device or another system. A person may play several roles, and several people may play one role. A primary actor seeks a goal; a supporting actor provides a service. Ask which user groups perform main functions, administer or maintain the system, and which external systems exchange information.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

> Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
> Use cases describe the behavior of the system as seen from an actor's point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
> use case to access the system functionality. The use case can then initiate other use cases and gather
> more information from the actors. When actors and use cases exchange information, they are said
> to communicate. We will see later that we represent these exchanges with communication relationships.

Software Engineering module | PDF page 62 | lines 9-16

Requirement's elicitation focuses on describing the purpose of the system. The client, the developers, and the users identify a problem area and define a system that addresses the problem.
Such a definition is called a system specification and serves as a contract between the client and the developers. The system specification is structured and formalized during analysis to produce an analysis model (see Figure 3.1). Both system specification and analysis model represent the same information. They differ only in the language and notation they use. The system specification
is written in natural language, whereas the analysis model is usually expressed in a formal or semiformal notation.

Question sources

Computer Science MCQ practice collection · Question 48

Question 213 | AM0049

What is an alternative flow in a use case?

A. A flowchart representation of the use case steps

B. A flow of activities that occurs in parallel with the main flow

C. A different sequence of steps based on specific conditions

D. A representation of the relationship between use cases

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: A different sequence of steps based on specific conditions

A flow of events describes interactions; exceptional conditions introduce alternative event sequences. This is a different path under a condition, rather than necessarily concurrency or inter-use-case structure.

Why each choice is right or wrong

A - Incorrect: A flowchart is a representation format, not the alternative sequence itself. See PDF p. 48, 50.

B - Incorrect: Parallel execution is not required by an alternative branch. See PDF p. 48, 50.

C - Correct: Directly describes a condition-dependent event sequence. See PDF p. 48, 50.

D - Incorrect: Include/extend/generalization describe relationships, not the definition of every alternative flow. See PDF p. 48, 50.

Primary note proof

Study-note section SE15

Example: Submit Incident. Precondition: officer authenticated. 1 Officer supplies location and description. 2 System validates details. 3 System records the incident. 4 System returns its ID. Alternative 2a: location missing → report the problem and request correction; save nothing. Success postcondition: incident and ID persist; failure guarantee: no partial record. An alternate flow is a conditional variation; an exception flow handles a failure. Preconditions hold before entry, while a trigger starts the use case.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 48 | lines 18-32

To describe a use case, we use a template composed of six fields (see also Figure 2.2):
• The name of the use case is unique across the system so that developers (and project participants) can unambiguously refer to the use case.
• Participating actors are actors interacting with the use case.
• Entry conditions describe the conditions that need to be satisfied before the use case is initiated.
• The flow of events describes the sequence of actions of the use case, which are be numbered for reference. The common case (i.e., cases that occur frequently) and the exceptional cases (i.e., cases that seldom occur, such as errors and unusual conditions) are described separately in different use cases for clarity.
• Exit conditions describe the conditions that are satisfied after the completion of the use case.
• Special requirements are requirements that are not related to the functionality of the system. These include constraints on the performance of the system, its implementation, the hardware platforms it runs on, and so on.

Software Engineering module | PDF page 50 | lines 14-24

Extend relationships
Extend relationships are an alternate means for reducing complexity in the use case model. A use
case can extend another use case by adding events. An extend relationship indicates that an instance of an extended use case may include (under certain conditions) the behavior specified by
the extending use case. A typical application of extend relationships is the specification of exceptional behavior. For example (Figure 2-19), assume that the network connection between the
Dispatcher and the FieldOfficer can be interrupted at any time. (e.g., if the FieldOfficer enters a tunnel). The use case ConnectionDown describes the set of events taken by the system and the actors while the connection is lost. ConnectionDown extends the use cases OpenIncident and AllocateResources. Separating exceptional behavior from common behavior enables us to write shorter and more focused use cases.

Question sources

Computer Science MCQ practice collection · Question 49

Question 214 | AM0050

What is the purpose of a use case diagram?

A. To describe the behavior of a use case in detail

B. To represent the internal structure of a use case

C. To visualize the relationships between actors and use cases

D. To represent the flow of activities within a use case

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: To visualize the relationships between actors and use cases

The diagram shows actors, functions and communication links; the separate textual template contains detailed event sequences and conditions.

Why each choice is right or wrong

A - Incorrect: Detailed flow belongs in the use-case description/template. See PDF p. 47, 48.

B - Incorrect: Internal structure is modeled with classes/components. See PDF p. 47, 48.

C - Correct: Direct high-level actor/function interaction view. See PDF p. 47, 48.

D - Incorrect: Activity diagrams or textual flows express detailed action sequences. See PDF p. 47, 48.

Primary note proof

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 47 | lines 30-39

> Actors are external entities that interact with the system. Examples of actors include a user role (e.g., a system administrator, a bank customer, a bank teller) or another system (e.g., a central database, a fabrication line). Actors have unique names and descriptions.
> Use cases describe the behavior of the system as seen from an actor's point of view. Behavior described by the use case model is also called external behavior. A use case describes a function provided by the system as a set of events that yields a visible result for the actors. Actors initiate a
> use case to access the system functionality. The use case can then initiate other use cases and gather

more information from the actors. When actors and use cases exchange information, they are said
to communicate. We will see later that we represent these exchanges with communication relationships.

Software Engineering module | PDF page 48 | lines 18-32

To describe a use case, we use a template composed of six fields (see also Figure 2.2):
• The name of the use case is unique across the system so that developers (and project participants) can unambiguously refer to the use case.
• Participating actors are actors interacting with the use case.
• Entry conditions describe the conditions that need to be satisfied before the use case is initiated.
• The flow of events describes the sequence of actions of the use case, which are be numbered for reference. The common case (i.e., cases that occur frequently) and the exceptional cases (i.e., cases that seldom occur, such as errors and unusual conditions) are described separately in different use cases for clarity.
• Exit conditions describe the conditions that are satisfied after the completion of the use case.
• Special requirements are requirements that are not related to the functionality of the system. These include constraints on the performance of the system, its implementation, the hardware platforms it runs on, and so on.

Question sources

Computer Science MCQ practice collection · Question 50

Question 215 | CS0007

Identifying and describing the fundamental software system abstractions and their relationships. Establishing the overall system architecture - in which phase are these activities done?

A. System design

B. Requirement analysis and definition

C. System testing

D. Implementation

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer A: System design

System and software design establish the overall architecture and identify software abstractions and relationships.

Why each choice is right or wrong

A - Correct: Direct match to the described design tasks. See PDF p. 18, 116.

B - Incorrect: Establishes required services, constraints and goals. See PDF p. 18, 116.

C - Incorrect: Checks complete-system behavior. See PDF p. 18, 116.

D - Incorrect: Translates a design into code. See PDF p. 18, 116.

Primary note proof

Study-note section SE21

System design turns the analysis model into architecture: design goals, subsystem responsibilities, dependencies, hardware mapping, data management, access control, control flow and startup/shutdown/error policies. Detail/object design refines APIs, parameter/return types, exceptions, component choices and algorithms. A subsystem is a manageable logical collection with services and an interface; it need not be a physical replaceable artifact.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 18 | lines 13-20

System and software design: The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture.
Software design involves identifying and describing the fundamental software system abstractions
and their relationships. The goal of the design phase is to transform the requirements specified in
the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the
SRS document. Two distinctly different approaches are available: the traditional design approach
and the object-oriented design approach.

Software Engineering module | PDF page 116 | lines 16-29

The analysis model describes the system completely from the actors' point of view and serves as the basis of communication between the client and the developers. The analysis model, however, does not contain information about the internal structure of the system, its hardware configuration,
or, more generally, how the system should be realized. System design is the first step in this direction. System design results in the following products:
• List of design goals, describing the qualities of the system that developers should optimize
• Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.
The design goals are derived from the nonfunctional requirements. Design goals guide the decisions to be made by the developers especially when trade-offs are needed. The subsystem decomposition constitutes the bulk of system design. Developers divide the system into

manageable pieces to deal with complexity: Each subsystem is assigned to a team and realized independently. In order for this to be possible, though, developers need to address system-wide

Question sources

Computer Science Exit Exam practice collection · Question 7

Question 216 | CS0008

Instead of delivering the system as a single delivery, development and delivery is broken down into
phases with each phase delivering part of the required functionality. What type of development model is this?

A. Incremental

B. Waterfall

C. Spiral

D. Evolutionary

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer A: Incremental

Successive deliveries that each add part of the required usable functionality define incremental development. The product gains capability across these deliveries rather than waiting for a single completed-system handoff. Waterfall and a generic linear model organize phase sequence. Spiral is organized around risk assessment, and evolutionary development broadly describes refinement. Those can overlap with staged delivery, but incremental is the specific name for the delivery organization stated.

Why each choice is right or wrong

A - Correct: Incremental development adds usable required functionality in successive deliveries.

B - Incorrect: Waterfall names planned phase progression, not this particular staged-capability strategy.

C - Incorrect: Spiral names risk-driven cycles, even when increments are produced.

D - Incorrect: Evolutionary refinement is broader than the specific addition-of-functionality strategy described.

Primary note proof

Study-note section SE04

Choose incremental delivery when prioritized partial capability must reach users early. Smaller changes can be easier to debug and permit feedback, though architecture and integration still

require planning. Iteration need not rebuild the whole system every time, and increments need not be perfectly independent modules. Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Study-note section SE06

Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 22 | lines 8-12

This model delivers software in small but usable pieces, called —increments. In general, each increment builds on those that have already been delivered. The model is iterative in nature and focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and
also provide a platform for evaluation by the user.

Question sources

Computer Science Exit Exam practice collection · Question 8

Question 217 | CS0009

We should NOT use the Waterfall software development process model in situations like:

A. Plenty of resources with required expertise are available freely

B. There are no ambiguous requirements

C. The project is short

D. Software requirements change quickly

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer D: Software requirements change quickly

Waterfall should be used only when requirements are well understood and unlikely to change radically. Rapid requirement change violates this selection condition.

Why each choice is right or wrong

A - Incorrect: Available expertise is not the documented contraindication. See PDF p. 20.

B - Incorrect: Unambiguous stable requirements support, rather than undermine, the model. See PDF p. 20.

C - Incorrect: The module says waterfall works well for smaller projects with well-understood requirements. See PDF p. 20.

D - Correct: Directly violates the stability assumption and creates costly rework. See PDF p. 20.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 20 | lines 18-22

> the development plan. Its major problem is the inflexible partitioning of the project into distinct stages. Commitments must be made at an early stage in the process, which makes it difficult to respond to changing customer requirements.
> In principle, the waterfall model should only be used when the requirements are well understood
> and unlikely to change radically during system development. However, the waterfall model

Question sources

Computer Science Exit Exam practice collection · Question 9

Question 218 | CS0010

Which one of the following is false about divide and conquer or modularization process?

A. Separate people can work on each sub-divided part

B. Opportunities arise for making the component reusable

C. Each individual component is smaller, and therefore easier to understand

D. Replacing or changing a component is difficult because it affects other parts extremely

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: Replacing or changing a component is difficult because it affects other parts extremely

Subsystems can be assigned to separate teams, reused and changed with reduced impact when coupled loosely. D contradicts the change-isolation goal of good modularization.

Why each choice is right or wrong

A - Incorrect: Independent teams can realize different manageable subsystems. See PDF p. 118, 116, 32, 120.

B - Incorrect: Reusable self-contained modules are an explicit OO benefit. See PDF p. 118, 116, 32, 120.

C - Incorrect: Decomposition produces simpler manageable units. See PDF p. 118, 116, 32, 120.

D - Correct: Strong propagation of every change describes undesirable tight coupling, not good modularization. See PDF p. 118, 116, 32, 120.

Primary note proof

Study-note section SE22

> Logical cohesion groups similar-category operations, often selected by a flag; temporal cohesion groups operations done at the same time; functional cohesion focuses on one task. Splitting an unrelated three-purpose component improves modularity. If behavior is preserved, the restructuring is refactoring; the words are not mutually exclusive answers. Smaller modules can aid understanding, independent work, reuse and maintenance, but too many interfaces also create complexity.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 118 | lines 11-16

> 6.1.1. Subsystems and classes
> In order to reduce the complexity of the application domain, we identified smaller parts called classes and organized them into packages. Similarly, to reduce the complexity of the solution domain, we decompose a system into simpler parts, called subsystems, which are made of a number of solution domain classes. In the case of complex subsystems, we recursively apply this principle and decompose a subsystem into simpler subsystems (see Figure 6.2).

Software Engineering module | PDF page 116 | lines 16-29

> The analysis model describes the system completely from the actors' point of view and serves as the basis of communication between the client and the developers. The analysis model, however, does not contain information about the internal structure of the system, its hardware configuration,
> or, more generally, how the system should be realized. System design is the first step in this direction. System design results in the following products:
> • List of design goals, describing the qualities of the system that developers should optimize
> • Software architecture, describing the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems, subsystem mapping to hardware, and major policy decisions such as control flow, access control, and data storage.
> The design goals are derived from the nonfunctional requirements. Design goals guide the decisions to be made by the developers especially when trade-offs are needed. The subsystem decomposition constitutes the bulk of system design. Developers divide the system into manageable pieces to deal with complexity: Each subsystem is assigned to a team and realized independently. In order for this to be possible, though, developers need to address system-wide

Software Engineering module | PDF page 32 | lines 12-21

A class in an object-oriented system carefully delineates between its interfaces the routines and attributes within a class are held together tightly. In a properly designed system, the classes will be grouped into subsystems but remain independent; therefore, changing one class has no impact
on other classes, and so, the impact is minimized. However, the object-oriented approach is not a panacea; nothing is magical here that will promote perfect design or perfect code.
Promotion of reusability
Objects are reusable because they are modeled directly out of a real-world problem domain. Each
object stands by itself or within a small circle of peers (other objects). Within this framework, the class does not concern itself with the rest of the system or how it is going to be used within a particular system.

Software Engineering module | PDF page 120 | lines 6-12

6.1.3. Coupling and coherence
Coupling is the strength of dependencies between two subsystems. If two subsystems are loosely coupled, they are relatively independent, and thus, modifications to one of the subsystems will have little impact on the other. If two subsystems are strongly coupled, modifications to one subsystem is likely to have impact on the other. A desirable property of a subsystem decomposition
is that subsystems are as loosely coupled as possible. This minimizes the impact that errors or future changes have on the correct operation of the system.

Question sources

Computer Science Exit Exam practice collection · Question 10

Question 219 | CS0013

For this project, planned testing effort in hours is setup + (cases × hours per case). The estimates (setup,cases,hours/case) are Unit=(4,80,0.1), Acceptance=(3,10,1), System=(6,20,0.5), Integration=(4,20,0.5). Which level has the greatest planned effort?

A. Unit testing

B. Acceptance testing

C. System testing

D. Integration testing

Hint

Use the effort formula given in the question separately for every level, then compare all computed totals.

Answer C: System testing

The estimates give unit effort 4+80×0.1=12 hours, acceptance 3+10×1=13, system 6+20×0.5=16, and integration 4+20×0.5=14. System testing therefore has the greatest planned effort for this project. The note does not make any test level universally longest; the explicit workload and setup assumptions are what justify this particular ranking.

Why each choice is right or wrong

A - Incorrect: Unit totals twelve hours, below the maximum.

B - Incorrect: Acceptance totals thirteen hours, below the maximum.

C - Correct: System totals sixteen hours, the largest estimate.

D - Incorrect: Integration totals fourteen hours, below system testing.

Primary note proof

Study-note section SE35

> Unit testing isolates a function, class or small component, often replacing dependencies with stubs/mocks; a driver invokes the unit and a stub substitutes a called dependency. Integration testing exercises interfaces and interactions among combined parts. System testing checks the complete integrated system against functional and nonfunctional requirements in a representative environment.

Study-note section SE35

> None of these levels is universally the longest.

Study-note section SE28

> Planning is continuous: identify scope/deliverables, decompose work, estimate effort/duration/cost, assign resources, sequence dependencies and revise as evidence changes.

Module passages - related reading

These source passages provide context; the primary note proof is identified separately above.

Software Engineering module | PDF page 148 | lines 19-26

> Unit testing focuses on the building blocks of the software system, that is, objects and subsystems.
> There are three motivations behind focusing on components. First, unit testing reduces the complexity of the overall test activities, allowing us to focus on smaller units of the system. Second, unit testing makes it easier to pinpoint and correct faults, given that few components are involved in the test. Third, unit testing allows parallelism in the testing activities; that is, each component can be tested independently of one another.
> The specific candidates for unit testing are chosen from the object model and the system decomposition of the system. In principle, all the objects developed during the development

Software Engineering module | PDF page 144 | lines 17-32

> Figure 6.2 depicts an overview of the testing activities:
> • Unit testing tries to find faults in participating objects and/or subsystems with respect to

the use cases from the use case model
• Integration testing is the activity of finding faults when testing the individually tested components together, for example, subsystems described in the subsystem decomposition, while executing the use cases and scenarios from the RAD.
o The system structure testing is the culmination of integration testing involving all components of the system. Integration tests and structure tests exploit knowledge from the SDD using an integration strategy described in the Test Plan (TP).
• System testing tests all the components together, seen as a single system to identify errors with respect to the scenarios from the problem statement and the requirements and design goals identified in the analysis and system design, respectively:
o Functional testing tests the requirements from the RAD and, if available, from the user manual.
o Performance testing checks the nonfunctional requirements and additional design goals from the SDD. Note that functional and performance testing are both done by

Question sources

Computer Science Exit Exam practice collection · Question 13

Question 220 | NS0293

A filled composition diamond is at Car's end of its Engine association. In this model Engine remains owned for its entire lifetime and cannot be detached or transferred. What does this imply?

A. Car uses Engine temporarily

B. Engine can exist independently of the Car

C. Engine is part of Car and cannot exist independently under the stated lifetime policy.

D. Car inherits from Engine

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Engine is part of Car and cannot exist independently under the stated lifetime policy.

Composition makes Engine a strongly owned part of Car, with the filled diamond beside the whole. Parts still owned are destroyed when their composite is destroyed. Since this model does not permit Engine to detach or transfer, it cannot continue an independent lifetime without Car. Temporary use would be a dependency, and inheritance would be shown by a hollow triangle.

Why each choice is right or wrong

A - Incorrect: Temporary use is a dependency rather than composition.

B - Incorrect: The no-detachment policy excludes an independent Engine lifetime.

C - Correct: Engine is Car's owned part and cannot outlive it under this policy.

D - Incorrect: Inheritance uses a hollow triangle.

Primary note proof

Study-note section SE18

> Composition uses a filled diamond at the whole end and strong ownership: a part is in at most one composite at a time, and parts still owned are destroyed with that composite. Parts may be detached/transferred under a defined model. Thus "cannot ever exist independently" is an oversimplification. For Car composing Engine, the diamond belongs beside Car; saying it points to Engine is imprecise. The Car-Engine example describes composition.

Question sources

Computer Science Exit Exam · 2018 · Question 10

Question 221 | NS0328

From the following statement which one is disadvantage of waterfall model?

A. model is simple and easy to understand and use

B. It is easy manage due to the rigidity of the model

C. Not suitable for projects where requirements are at a moderate to high risk of changing

D. In this model phases are processed and completed one at a time

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Poor suitability when requirements have substantial change risk.

SE03 says waterfall fits stable understood requirements and familiar technology. Late feedback can make changes costly, so moderate-to-high requirement-change risk is a direct disadvantage. Simplicity and milestone visibility are advantages. Sequential phase organization characterizes the classical model; it can contribute to rigidity, but C explicitly identifies the adverse scenario asked for.

Why each choice is right or wrong

A - Incorrect: Simplicity and ease of understanding are presented as advantages rather than the adverse scenario.

B - Incorrect: Manageable milestones can be an advantage of planned rigidity, despite its trade-offs.

C - Correct: Likely requirement changes make late feedback and rigid commitments costly.

D - Incorrect: Sequential phases describe the model; C more directly states the requested disadvantage.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Question sources

Computer Science Exit Exam · 2018 · Question 55

Question 222 | NS0333

For a search service whose functions and database technology are already specified, which requirement adds a quantitative service-speed target?

A. The system allow user to generate PDF report.

B. The system respond to user's request within 2 second under normal load

C. The system allow user to store data on relational database

D. The system allow user to submit emergency report

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: The system respond to user's request within 2 second under normal load

The two-second response obligation bounds the speed of a service under the stated load, so it is a performance requirement. Generating a PDF and submitting an emergency report describe services the system provides. The database clause describes storage behavior together with an implementation choice; it does not impose a response-time threshold. The explicit performance scope removes the broader taxonomy overlap between nonfunctional requirements and technology constraints.

Why each choice is right or wrong

A - Incorrect: Report generation specifies a service, not its response-time limit.

B - Correct: A two-second response bound under normal load is a quantitative speed target.

C - Incorrect: This specifies storage behavior and a database technology, without a timing bound.

D - Incorrect: Submitting an emergency report specifies a function rather than its speed.

Primary note proof

Study-note section SE10

> Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning. Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Study-note section SE10

> "Search responds within two seconds under 100 concurrent users" is measurable performance. "New users finish checkout within one minute without help" is usability. "Use Java" is a technology/pseudo-requirement, often separately classified as a system constraint. Dependability includes reliability, safety and robustness; portability is not its usual central category. Performance degradation at peak load primarily concerns performance, not automatically reliability.

Question sources

Computer Science Exit Exam · 2018 · Question 61

Question 223 | NS0347

Which one of the following UML building block that defines the static part of the model and represents physical and conceptual elements?

A. Structural things

B. Behavioral things

C. An notational things

D. Grouping things

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer A: Structural things.

SE17 explicitly describes structural things as conceptual or physical structural elements. They make up the static model in contrast with behavior over time. Behavioral things cover dynamic aspects, grouping organizes elements and annotations add explanatory information rather than being the static elements themselves. Thus A directly matches both parts of the definition.

Why each choice is right or wrong

A - Correct: Structural things are the static conceptual or physical building blocks described.

B - Incorrect: Behavioral things concern dynamic behavior rather than the static structural elements.

C - Incorrect: Annotations add explanatory information rather than defining the static structural building blocks.

D - Incorrect: Grouping organizes model elements rather than being the stated conceptual/physical structural category.

Primary note proof

Study-note section SE17

> A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links. UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process. Structural things describe conceptual or physical structural elements.

Study-note section SE-S03

> In the course's UML building-block terminology, structural things describe static conceptual or physical elements. Behavioral things describe dynamic interactions and state behavior.

Grouping things organize model elements into packages. Annotational things attach explanatory remarks or constraints, usually as notes.

Question sources

Computer Science Exit Exam · 2018 · Question 80

Question 224 | NS0740

What is the difference between functional and non-functional requirements? Give an example of each.

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer: Functional requirements specify observable behavior; nonfunctional requirements specify qualities or constraints. Example: generate a PDF report versus respond within two seconds under a stated load.

SE09 includes required and prohibited services, so generating a PDF report is functional. SE10 specifies qualities such as performance, usability and availability; a response-time threshold under defined load is nonfunctional. Security mechanisms can implement functional behavior as well as support quality goals, so a negative or security-related statement is not automatically nonfunctional.

Primary note proof

Study-note section SE09

> A functional requirement states an observable service or behavior, including reactions to inputs, errors and forbidden actions. “When the sensor detects intrusion, send an alert” is functional.

Study-note section SE09

> “The system shall generate a PDF report” is functional. “The system shall not reveal another customer’s balance” can be a security-related functional rule. Security as a broad quality goal and the functions implementing it must be distinguished.

Study-note section SE10

> Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning.

Study-note section SE10

> “Search responds within two seconds under 100 concurrent users” is measurable performance. “New users finish checkout within one minute without help” is usability.

Question sources

Software Engineering review questions · Question 1

Question 225 | NS0741

Why is low coupling considered good in software design?

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer: Low coupling limits dependencies, so changes remain local and modules are easier to understand, replace, test and maintain.

SE22 measures coupling through dependencies between modules and recommends low coupling with high cohesion. Shared mutable data and knowledge of another module's representation increase the chance that one change affects others. Stable interfaces and information hiding constrain that impact. More interfaces can add complexity, so minimizing harmful dependencies matters more than simply maximizing the number of modules.

Primary note proof

Study-note section SE22

> Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module's internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Question sources

Software Engineering review questions · Question 2

Question 226 | NS0742

What is the difference between Waterfall and Agile methodologies?

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer: Waterfall uses planned, reviewed stages and suits stable requirements; Agile emphasizes collaboration, working results, feedback and adaptation.

SE03 describes planned phases with costly late feedback, useful for stable requirements and familiar technology. SE05 values customer participation and adaptation with useful working increments. Agile still plans and documents; waterfall can revisit work under control. The practical difference is feedback and commitment organization, not a claim that one process forbids change or the other forbids documentation.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Study-note section SE05

> Agile favors people collaborating, useful working results, customer participation and adaptation over rigid adherence to procedures, exhaustive documents, contracts or fixed plans. The latter still have value. Agile does not mean no planning or no documentation. A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Question sources

Software Engineering review questions · Question 3

Question 227 | NS0743

What is the difference between <<include>> and <<extend>> in use case diagrams?

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer: Include factors required reused behavior, with base→included; extend adds conditional behavior, with extension→base.

SE16 uses include for reused required behavior incorporated when the inclusion point is reached. Extend adds behavior at extension points under a condition, and the base remains meaningful independently. The dashed arrows have different directions: including use case toward included, and extending use case toward base. Neither relation should be confused with actor association or specialization generalization.

Primary note proof

Study-note section SE16

> Include factors required reused behavior: Submit Order includes Validate Order, with the dashed arrow from the including/base use case to the included one. Extend adds conditional behavior at extension points: Apply Discount extends Checkout, with the arrow from the extension to the base. The base is meaningful independently. For an extension, draw the arrow from the extending use case to the base use case.

Study-note section SE16

> Generalization relates a specialized actor/use case to a more general one, with a hollow triangle toward the general element. An abstract use case specifies behavior for specialization and is not directly instantiated as a complete use case. It is not defined merely by reuse or lack of a direct actor line. Included behavior is incorporated where the inclusion is reached; do not infer every conditional branch in the base always executes.

Question sources

Software Engineering review questions · Question 4

Question 228 | NS0744

Why is maintenance often the most expensive phase of the SDLC?

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer: Maintenance can dominate cost because it lasts years and repeatedly requires legacy understanding, changes, regression testing and release work.

SE40 says maintenance often spans years and includes understanding existing code plus regression effort. Work is not limited to bug fixes: environments, useful capabilities and future risks also change. Repeated impact analysis, implementation and testing accumulate across the operational lifetime, so maintenance may dominate lifecycle cost. This is a common outcome, not a guarantee for every project.

Primary note proof

Study-note section SE40

> Corrective maintenance fixes faults; adaptive maintenance responds to environment changes; perfective maintenance improves useful capability/performance; preventive maintenance reduces future risk, such as restructuring fragile code. Motive determines category; refactoring can support preventive or perfective work. Maintenance often lasts years and includes understanding legacy code and regression effort, so it may dominate lifecycle cost.

Study-note section SE40

> A typical evolution sequence is change request→impact analysis→release planning→implementation/testing→release. Among a list lacking the request step, impact analysis may be the earliest listed technical activity. Major restructuring is reengineering; if the architecture itself changes, architectural transformation is the more specific description. Markets, technology, stakeholders and funding can all change scope or priorities.

Question sources

Software Engineering review questions · Question 5

Question 229 | NS0745

Which software development approach emphasizes breaking systems into functions or procedures that manipulate isolated data?

A. Object-Oriented Approach

B. Traditional Approach

C. Agile Approach

D. Iterative Approach

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Traditional/procedural decomposition.

SE01 contrasts procedural decomposition organized around functions with object-oriented decomposition organized around objects combining state and behavior. B is the available traditional/procedural label for functions operating on data. Agile and iterative describe lifecycle organization rather than this decomposition paradigm, and either can be used with procedural or object-oriented programming.

Why each choice is right or wrong

A - Incorrect: Object orientation combines state and behavior in objects rather than organizing primarily around separate procedures.

B - Correct: Traditional/procedural decomposition organizes functions and procedures acting on data.

C - Incorrect: Agile concerns process adaptation rather than requiring function-based decomposition.

D - Incorrect: Iteration describes revisiting development work and does not determine procedural versus object decomposition.

Primary note proof

Study-note section SE01

> Procedural decomposition organizes functions; object oriented decomposition organizes objects with state and behavior. Either can use iterative or sequential development. OOP does not automatically shorten every project. Reuse, encapsulation and consistent models can reduce duplication and ease change; poor object design can still be complex. Functional programming emphasizes function evaluation and controlled side effects; event-driven programs react to events.

Question sources

Software Engineering review questions · Question 1

Question 230 | NS0746

In the Object-Oriented Approach, which property allows classes to be used in different parts of the system or other projects without rewriting them?

A. Encapsulation

B. Polymorphism

C. Reusability

D. Abstraction

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Reusability.

The stem explicitly asks about using existing classes elsewhere without rewriting them, which is reuse. SE01 lists reuse alongside encapsulation and consistent models as ways to reduce duplication and ease change. Encapsulation and abstraction can make reuse safer, while polymorphism supports compatible varying implementations, but those are enabling concepts rather than the name of the capability described.

Why each choice is right or wrong

A - Incorrect: Encapsulation hides implementation and can enable reuse, but it is not the capability's name.

B - Incorrect: Polymorphism supports differing implementations through compatible interfaces rather than directly naming reuse.

C - Correct: Reusability means using suitable existing classes or components in other contexts without rewriting them.

D - Incorrect: Abstraction exposes relevant contracts and can assist reuse, but the requested property is reusability.

Primary note proof

Study-note section SE-S02

> Encapsulation groups state with operations and hides representation behind an interface. Polymorphism permits different object types to respond differently through a common operation or interface. Abstraction exposes the relevant contract while omitting unnecessary detail. Reusability means using suitable existing classes or components in new contexts rather than writing them again. These ideas support each other but name different properties.

Question sources

Software Engineering review questions · Question 2

Question 231 | NS0747

Which of the four fundamental software process activities involves making updates to meet changing customer needs after deployment?

A. Software Specification

B. Software Design and Implementation

C. Software Validation

D. Software Evolution

Hint

Classify the motive for the change before its implementation: repair, environment, useful capability or reduced future risk.

Answer D: Software evolution.

SE40 describes maintenance and evolution through change requests, impact analysis, planning, implementation/testing and release. Updating a deployed product to meet changing customer needs belongs to evolution among the four broad process activities. Specification, implementation and validation still occur within that work, but D names the lifecycle-wide change activity asked for rather than one constituent task.

Why each choice is right or wrong

A - Incorrect: Specification captures needs but does not name the overall postdeployment change activity.

B - Incorrect: Design/implementation is part of making a change rather than the broad evolution activity.

C - Incorrect: Validation checks the changed product against needs rather than naming ongoing adaptation itself.

D - Correct: Evolution updates existing software as stakeholder needs and operating conditions change.

Primary note proof

Study-note section SE40

> Corrective maintenance fixes faults; adaptive maintenance responds to environment changes; perfective maintenance improves useful capability/performance; preventive maintenance reduces future risk, such as restructuring fragile code. Motive determines category; refactoring can support preventive or perfective work. Maintenance often lasts years and includes understanding legacy code and regression effort, so it may dominate lifecycle cost.

Study-note section SE40

> A typical evolution sequence is change request→impact analysis→release planning→implementation/testing→release.

Question sources

Software Engineering review questions · Question 3

Question 232 | NS0748

A software process that plans all activities in advance and measures progress against that plan is classified as:

A. Agile Process

B. Plan-Driven Process

C. Incremental Process

D. Evolutionary Process

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Plan-driven process.

The stem defines up-front planning and measurement against that plan. B names the plan-driven organization, illustrated by SE03's planned phases and reviewed deliverables. Incremental delivery and evolution describe how results grow or change and can coexist with planning. Agile also plans, but it emphasizes adaptation rather than defining all activity commitments in advance as stated here.

Why each choice is right or wrong

A - Incorrect: Agile uses planning but adapts content as feedback arrives rather than defining this fixed up-front emphasis.

B - Correct: Plan-driven explicitly organizes activities and progress measurement around an advance plan.

C - Incorrect: Incremental describes staged capability delivery and does not alone specify up-front planning commitments.

D - Incorrect: Evolutionary describes progressive change rather than necessarily advance planning of all activities.

Primary note proof

Study-note section SE-S04

> In a plan-driven process, activities are planned in advance and progress is assessed against that plan. Agile planning is incremental and adapts to changing needs. This distinction concerns planning strategy; a process can combine advance planning with staged deliveries and feedback.

Question sources

Software Engineering review questions · Question 4

Question 233 | NS0749

In requirement elicitation, the document created to describe what the system should do - agreed upon by clients and developers - is called:

A. Analysis Model

B. Use Case Diagram

C. Requirements Specification

D. Design Document

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Requirements Specification.

SE08 states that the Software Requirements Specification organizes agreed requirements. That is the artifact describing what the system should do for client/developer agreement. Analysis models support technical discussion, a use-case diagram captures only selected goals/relationships and a design document describes solution decisions. None replaces the overall agreed requirements specification.

Why each choice is right or wrong

A - Incorrect: Analysis models formalize selected aspects for discussion rather than replacing the full agreed requirements document.

B - Incorrect: A use-case diagram shows goals and roles but not the complete requirements specification.

C - Correct: The requirements specification organizes the agreed services and constraints the system must satisfy.

D - Incorrect: A design document describes how the solution is organized rather than the complete statement of required behavior.

Primary note proof

Study-note section SE08

> Business requirements explain why the project exists and its goals. User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements. A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Question sources

Software Engineering review questions · Question 5

Question 234 | NS0750

What is the key distinction between a Requirements Specification and an Analysis Model?

A. The specification uses diagrams; the model uses natural language

B. The specification uses natural language; the model uses symbols and diagrams

C. Both use the same representation but serve different purposes

D. The analysis model is only for developers; specification is for testing

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: In the taught convention, natural-language client specification versus symbolic/diagrammatic analysis models.

SE08 explicitly describes natural-language specification for clients and analysis models that formalize it for technical discussion. B matches that teaching convention. This is not a universal ban on diagrams in an SRS: the note immediately allows them in real specifications. The other choices reverse the convention or impose unsupported exclusive audiences and purposes.

Why each choice is right or wrong

A - Incorrect: It reverses the natural-language specification versus formalized-model convention taught in the note.

B - Correct: This matches the taught convention, while real requirements specifications can also include diagrams.

C - Incorrect: The taught distinction does use different representation emphases rather than identical forms.

D - Incorrect: Neither artifact has the exclusive audience or testing-only purpose asserted here.

Primary note proof

Study-note section SE08

> Business requirements explain why the project exists and its goals. User requirements describe stakeholder needs at a useful high level; system requirements specify detailed services and constraints. The Software Requirements Specification (SRS) organizes the agreed requirements. A natural-language specification communicates with clients while analysis models formalize it for technical discussion; real specifications can also contain diagrams.

Question sources

Software Engineering review questions · Question 6

Question 235 | NS0751

In use case modeling, actors can be defined as:

A. Only human users who interact with the system

B. People, machines, or external systems that interact with the system

C. Only primary users who need system assistance

D. Software components within the system boundary

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: External interacting roles played by people, devices or other systems.

SE14 explicitly permits a person, device or other system to play an external actor role. B is the only option including those categories. Actors are not restricted to primary users or humans, and internal components are inside the chosen system boundary rather than external roles. One person may play several actors, so actor instances should not be confused with individual people.

Why each choice is right or wrong

A - Incorrect: Actors can be devices or external systems as well as human users.

B - Correct: People, devices and external systems can play roles interacting across the system boundary.

C - Incorrect: Supporting and administrative external roles can be actors, not only primary goal-seeking users.

D - Incorrect: Internal components are not external actor roles under the chosen system boundary.

Primary note proof

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Study-note section SE14

> An actor is an external role interacting with the system; it can be a person, device or another system.

Question sources

Software Engineering review questions · Question 7

Question 236 | NS0752

Which type of requirement specifies that 'the system must respond within 1 second'?

A. Functional Requirement

B. System Constraint

C. Non-Functional Requirement

D. Domain Requirement

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Nonfunctional performance requirement.

SE10 identifies response time as a performance quality. A one-second threshold constrains how well a service responds rather than naming the service itself, so C is the appropriate category. A complete measurable requirement should also define the operation, load and timing conditions. Those omissions affect test precision, not the underlying classification of the stated response-time target.

Why each choice is right or wrong

A - Incorrect: The statement constrains response speed rather than specifying which service is performed.

B - Incorrect: The specific quality is performance; this is not primarily a technology or implementation constraint.

C - Correct: The one-second limit is a nonfunctional response-time/performance target.

D - Incorrect: No domain-specific business rule is stated; the obligation concerns a general performance quality.

Primary note proof

Study-note section SE10

> Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning. Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Question sources

Software Engineering review questions · Question 8

Question 237 | NS0753

When technology constraints are separated from quality attributes, what is the most specific category for “the system must be built using Java”?

A. Functional Requirement

B. Non-Functional Requirement

C. System/Constraint Requirement

D. Quality Requirement

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: System/Constraint Requirement

The statement prescribes an implementation technology rather than an externally observable service or a measurable product quality. In the course’s explicit separation of quality attributes from technology constraints, the specific category is a system or constraint requirement. A broad requirements taxonomy can place constraints under nonfunctional requirements, but the question asks for the more specific category. No performance, reliability or usability threshold is supplied by merely naming Java.

Why each choice is right or wrong

A - Incorrect: The statement prescribes technology rather than an observable service.

B - Incorrect: This is a broader umbrella in some taxonomies; it is incomplete when the specific constraint category is offered.

C - Correct: A required implementation language is a technology or system constraint.

D - Incorrect: No measurable quality attribute is specified merely by requiring Java.

Primary note proof

Study-note section SE10

> “Search responds within two seconds under 100 concurrent users” is measurable performance. “New users finish checkout within one minute without help” is usability. “Use Java” is a technology/pseudo-requirement, often separately classified as a system constraint. Dependability includes reliability, safety and robustness; portability is not its usual central category. Performance degradation at peak load primarily concerns performance, not automatically reliability.

Question sources

Software Engineering review questions · Question 9

Question 238 | NS0754

Which nonfunctional requirement category addresses the ability of the system to work across different web browsers and devices?

A. Reliability

B. Scalability

C. Usability

D. Compatibility

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: Compatibility.

SE10 distinguishes compatibility, which concerns cooperating environments, from portability, which concerns moving environments. Supporting different browsers and devices in the question is the offered compatibility category. Reliability concerns failure-free service, scalability concerns growth and usability concerns task success/ease of learning. None of those names the stated cross-environment cooperation requirement as directly.

Why each choice is right or wrong

A - Incorrect: Reliability concerns failure-free service rather than which browsers and devices can cooperate with the system.

B - Incorrect: Scalability concerns handling increased workload rather than cross-browser/device compatibility.

C - Incorrect: Usability concerns ease of learning and successful tasks, not the supported environment set itself.

D - Correct: Compatibility addresses operating/cooperating across the specified browsers and device environments.

Primary note proof

Study-note section SE10

> Nonfunctional requirements include quality attributes and constraints. Performance concerns response time, throughput and resource use; reliability concerns failure-free service; availability concerns being ready for use; usability concerns task success and ease of learning. Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Study-note section SE-S07

> Scalability is the ability to accommodate additional workload while maintaining the required service, often by increasing available resources. It concerns growth in demand. Compatibility concerns supported cooperating environments, such as browsers and devices; usability concerns successful and learnable user tasks.

Question sources

Software Engineering review questions · Question 10

Question 239 | NS0755

Dependability in software engineering is a broader concept that includes all of the following EXCEPT:

A. Reliability

B. Robustness

C. Safety

D. Portability

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer D: Portability is the exception to the usual central dependability categories.

SE10 explicitly includes reliability, safety and robustness in dependability and says portability is not its usual central category. Portability instead concerns moving environments. Thus D is the listed exception. The distinction does not imply portability never affects practical service outcomes, only that it is a different quality category from the taught dependability grouping.

Why each choice is right or wrong

A - Incorrect: Reliability is explicitly included in the taught dependability grouping.

B - Incorrect: Robustness is explicitly included in the broader dependability concept.

C - Incorrect: Safety is explicitly included among dependability concerns.

D - Correct: Portability concerns moving environments rather than the usual central dependability categories.

Primary note proof

Study-note section SE10

> Dependability includes reliability, safety and robustness; portability is not its usual central category.

Study-note section SE10

> Security concerns protection; compatibility concerns cooperating environments; portability concerns moving environments; maintainability/supportability concern change, diagnosis and evolution.

Study-note section SE-S07

> Robustness concerns reliable operation across varied or difficult operating conditions and controlled failure outside the supported range.

Question sources

Software Engineering review questions · Question 11

Question 240 | NS0756

In requirements elicitation activities, which step involves formalizing scenarios into structured descriptions of system interactions?

A. Identifying Actors

B. Identifying Use Cases

C. Identifying Scenarios

D. Refining Use Cases

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Identifying use cases by abstracting concrete scenarios.

SE13 calls a scenario one concrete execution and a use case an abstraction of related success and alternative scenarios. Formalizing those interactions into the more general use-case description therefore identifies use cases. Identifying actors establishes external roles, identifying scenarios captures individual executions and later refinement adds detail to an already identified use case.

Why each choice is right or wrong

A - Incorrect: Actor identification establishes external roles rather than generalizing scenario executions into interaction descriptions.

B - Correct: Use cases abstract related concrete scenarios into structured goal-oriented interaction descriptions.

C - Incorrect: Scenario identification captures concrete executions rather than their use-case abstraction.

D - Incorrect: Refinement elaborates an existing use case; the scenario-to-use-case abstraction is the identification step asked here.

Primary note proof

Study-note section SE13

> A scenario is one concrete execution; a use case abstracts the related successful and alternative scenarios. Analysis integrates functional (use cases/scenarios), structural/object (classes) and dynamic (states/interactions) models. Entity objects represent domain information such as Incident; boundary objects mediate interaction, such as ReportForm; control objects coordinate a use case. A UI form is normally boundary, not entity.

Question sources

Software Engineering review questions · Question 12

Question 241 | NS0757

According to the Simple Use Case Writing Guide, which of the following is CORRECT practice?

A. Use passive voice so responsibility is unclear

B. Describe user interface elements in detail

C. Document exceptions separately from the main flow

D. Combine multiple tasks into one use case for brevity

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Document exceptions separately from the main success flow.

SE15 instructs writers to use active subject-verb steps and separate errors from normal flow. C follows that rule directly. Passive voice hides responsibility, detailed UI design distracts from goal-level behavior, and combining unrelated tasks does not produce a clear use case. The distinction concerns organizing scenarios rather than omitting exception handling.

Why each choice is right or wrong

A - Incorrect: Passive voice makes responsibility less clear, contrary to the active subject-verb guideline.

B - Incorrect: Detailed UI elements are not the goal-level behavior and flow structure emphasized by the guide.

C - Correct: Separating exception behavior keeps the main success flow clear while still documenting failures.

D - Incorrect: Combining multiple unrelated tasks sacrifices a coherent goal and consistent abstraction level.

Primary note proof

Study-note section SE15

> Example: Submit Incident. Precondition: officer authenticated. 1 Officer supplies location and description. 2 System validates details. 3 System records the incident. 4 System returns its ID. Alternative 2a: location missing → report the problem and request correction; save nothing. Success postcondition: incident and ID persist; failure guarantee: no partial record. An alternate flow is a conditional variation; an exception flow handles a failure. Preconditions hold before entry, while a trigger starts the use case.

Study-note section SE15

> Use active subject-verb steps at one abstraction level; separate errors from the normal flow.

Question sources

Software Engineering review questions · Question 13

Question 242 | NS0758

In the FAST (Facilitated Application Specification Technique), participants are initially asked to individually create lists of three item categories. Which is NOT one of them?

A. Things that are part of the environment surrounding the system

B. Things the system produces or outputs

C. Technical risks and constraints

D. Things the system uses or takes as inputs

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Technical risks and constraints are not one of the three initial FAST lists.

SE12 explicitly lists environment elements, outputs and inputs as the initial participant categories. A, B and D match them. Technical risks and constraints can matter elsewhere in engineering, but they are not one of those three named initial FAST lists. The negative question asks about that procedure, not whether risk information is ever useful.

Why each choice is right or wrong

A - Incorrect: Environment elements are explicitly one of the initial FAST categories.

B - Incorrect: System outputs are explicitly included in the three initial lists.

C - Correct: Risks and constraints are not one of the specified environment/output/input categories.

D - Incorrect: Inputs and things the system uses are part of the stated initial categories.

Primary note proof

Study-note section SE12

> In the taught FAST technique, participants list environment elements, outputs and inputs, consolidate lists and develop smaller specifications. QFD prioritizes customer value: normal needs are explicitly stated, expected needs are assumed, and exciting needs provide unanticipated benefit. Brainstorm first to generate ideas, then evaluate and prioritize; do not confuse that with letting every idea become a committed requirement.

Question sources

Software Engineering review questions · Question 14

Question 243 | NS0759

In Quality Function Deployment (QFD), which category of requirements represents features that go beyond customer expectations and provide great satisfaction?

A. Normal Requirements

B. Expected Requirements

C. Exciting Requirements

D. Implicit Requirements

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Exciting requirements.

SE12 distinguishes explicitly stated normal needs, assumed expected needs and exciting needs that bring unanticipated benefit. Features exceeding expectations and producing additional satisfaction match the exciting category. Calling them merely implicit would confuse customer assumptions with unexpected added value rather than identify the QFD term.

Why each choice is right or wrong

A - Incorrect: Normal requirements are explicitly stated needs rather than unexpected extra benefits.

B - Incorrect: Expected requirements are assumed by customers and do not denote surprising added satisfaction.

C - Correct: Exciting requirements deliver unanticipated benefit beyond the customer's expectations.

D - Incorrect: Implicit describes assumed needs more generally, not the QFD category for unanticipated benefits.

Primary note proof

Study-note section SE12

> In the taught FAST technique, participants list environment elements, outputs and inputs, consolidate lists and develop smaller specifications. QFD prioritizes customer value: normal needs are explicitly stated, expected needs are assumed, and exciting needs provide unanticipated benefit. Brainstorm first to generate ideas, then evaluate and prioritize; do not confuse that with letting every idea become a committed requirement.

Question sources

Software Engineering review questions · Question 15

Question 244 | NS0760

Which validation property ensures that the requirements specification does not contain contradictory statements?

A. Completeness

B. Consistency

C. Clarity

D. Correctness

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Consistency.

SE11 directly defines consistency as excluding contradictions. Completeness covers agreed scope, clarity gives one intended interpretation, and correctness reflects stakeholder needs. A specification can be complete yet inconsistent, or clear yet wrong, so these properties cannot substitute for the precise no-contradictions condition.

Why each choice is right or wrong

A - Incorrect: Completeness concerns covering scope and behavior, not whether requirements contradict each other.

B - Correct: Consistency requires the specification’s statements to coexist without contradiction.

C - Incorrect: Clarity addresses interpretive ambiguity rather than logical compatibility between requirements.

D - Incorrect: Correctness concerns actual stakeholder needs, a different validation property.

Primary note proof

Study-note section SE11

> Correct requirements reflect actual stakeholder needs. Complete requirements address the agreed scope, normal and exceptional behavior. Consistency excludes contradictions. Clarity/unambiguity gives one intended interpretation; feasibility/realism means achievable within constraints. Verifiability means objective evidence can check fulfillment. Replace “fast for most users” with a defined metric, workload and threshold.

Question sources

Software Engineering review questions · Question 16

Question 245 | NS0761

The Waterfall model is MOST appropriate when:

A. Requirements are expected to change frequently during development

B. The project is large and complex with evolving needs

C. Requirements are constant, the project is short, and technology is stable

D. The team needs to deliver working software in small increments

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Stable requirements and familiar, stable technology.

SE03 says classical waterfall fits well-understood stable requirements and familiar technology, directly supporting C among these alternatives. Frequent change, evolving needs and early small deliveries call for feedback or incremental capability rather than waterfall's late feedback. Short duration alone is not a proof, but C also supplies the decisive stability conditions stated in the note.

Why each choice is right or wrong

A - Incorrect: Frequent requirement changes make late-feedback sequential planning a poor fit.

B - Incorrect: Large evolving needs do not establish the stable requirements that favor classical waterfall.

C - Correct: Constant requirements and stable technology match the note's stated waterfall selection conditions.

D - Incorrect: Prioritized usable small deliveries are the rationale for incremental delivery rather than a single sequential release.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Question sources

Software Engineering review questions · Question 17

Question 246 | NS0762

Which is a characteristic drawback of classical waterfall compared with approaches that seek repeated early feedback?

A. Requires skilled developers

B. Late feedback can make requirement changes costly through rework of earlier phase deliverables.

C. No clear project milestones

D. Does not support documentation

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Late feedback can make requirement changes costly through rework of earlier phase deliverables.

Classical waterfall separates planned phases and may receive important feedback late. Correcting earlier assumptions can then require costly rework across completed deliverables, which is its characteristic drawback here. Changes are still possible under control. Skilled developers are needed across approaches, and waterfall provides milestone visibility and documentation, so the other alternatives do not identify the stated comparative disadvantage.

Why each choice is right or wrong

A - Incorrect: Skill requirements are not unique to waterfall.

B - Correct: Late feedback can make requirement changes costly after earlier phases.

C - Incorrect: Waterfall provides clear reviewed milestones.

D - Incorrect: Waterfall supports planned documentation.

Primary note proof

Study-note section SE03

> Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly. It does not make change physically impossible: real projects revisit work under control, and modified waterfall explicitly includes feedback. Rigidity offers milestone visibility but can freeze incorrect assumptions.

Question sources

Software Engineering review questions · Question 18

Question 247 | NS0763

What is the key conceptual distinction between iteration and incremental delivery, even when a process combines both?

A. Iterative adds new features each cycle; Incremental refines the same set repeatedly

B. Iteration revisits and improves a solution; an increment adds usable functionality.

C. They are identical; the terms are interchangeable

D. Incremental involves risk analysis; Iterative does not

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Iteration revisits and improves a solution; an increment adds usable functionality.

Iteration revisits and improves a solution, while an increment adds usable functionality. A process can refine existing capability and add new capability in the same cycle, so the concepts are compatible but not identical. Neither requires rebuilding the entire system or making every increment completely independent. Risk analysis is a separate concern rather than the structural distinction defining these two terms.

Why each choice is right or wrong

A - Incorrect: This reverses refinement and added functionality.

B - Correct: Iteration refines; increments add usable capability.

C - Incorrect: The terms differ even when combined in one process.

D - Incorrect: Risk analysis does not define their distinction.

Primary note proof

Study-note section SE04

> Choose incremental delivery when prioritized partial capability must reach users early. Smaller changes can be easier to debug and permit feedback, though architecture and integration still require planning. Iteration need not rebuild the whole system every time, and increments need not be perfectly independent modules. Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Study-note section SE04

> Iteration revisits and improves a solution; an increment adds usable functionality. A process may do both.

Question sources

Software Engineering review questions · Question 19

Question 248 | NS0764

A large application already has its core capabilities. The main development goal is to revisit and improve that existing solution repeatedly as feedback and future changes arrive, rather than primarily add new capability in each delivery. Which concept most directly names that emphasis?

A. Waterfall Model

B. V-Model

C. Iterative Model

D. Incremental Model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Iterative Model

The stated emphasis is iterative: revisiting and improving an existing solution in response to feedback. Incremental development names the addition of usable functionality, which is not the primary goal specified, although both can coexist. Classical waterfall's stable staged planning and the V-model's paired assurance structure do not directly name this refinement emphasis. Large size and anticipated change alone would not have selected a unique model.

Why each choice is right or wrong

A - Incorrect: Classical waterfall does not directly name repeated refinement as the organizing emphasis.

B - Incorrect: The V-model emphasizes paired development and tests.

C - Correct: Iteration directly names revisiting and improving the existing solution.

D - Incorrect: Incremental describes added functionality rather than the specified primary emphasis.

Primary note proof

Study-note section SE04

> Choose incremental delivery when prioritized partial capability must reach users early. Smaller changes can be easier to debug and permit feedback, though architecture and integration still require planning. Iteration need not rebuild the whole system every time, and increments need not be perfectly independent modules. Broad descriptions of early working software can fit prototypes, iterative or Agile processes; identify the purpose of that early version.

Study-note section SE02

> A common lifecycle names requirements, design, implementation, testing, deployment and maintenance. Analysis clarifies the problem; system design chooses architecture; object/detail design specifies interfaces and algorithms; implementation writes code; testing evaluates behavior; deployment puts it into service. These activities may overlap and repeat.

Study-note section SE04

> Iteration revisits and improves a solution; an increment adds usable functionality. A process may do both.

Question sources

Software Engineering review questions · Question 20

Question 249 | NS0765

In the Agile SDLC, small units of functionality that the software needs to deliver are called:

A. Sprints

B. Iterations

C. User Stories

D. Epics

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: User stories.

SE05 defines a user story as a small user need and value supported by acceptance criteria. Sprints and iterations name cycles or timeboxes, not the need itself. An epic groups broader work. C therefore names the small functional work unit intended here, without claiming that Agile requirements consist only of story text.

Why each choice is right or wrong

A - Incorrect: A Sprint is a timeboxed development interval rather than the individual user need.

B - Incorrect: An iteration is a cycle of work or refinement, not a small functional requirement.

C - Correct: A user story states a small user need and its value with acceptance criteria.

D - Incorrect: An epic groups broader work rather than naming the smaller story-sized need.

Primary note proof

Study-note section SE05

> Scrum uses timeboxed Sprints of at most one month, with a Sprint Goal, selected work and a usable Increment meeting a Definition of Done. Scope is discussed before and during planning; it is not forbidden to agree scope. Timeboxes tend to stay fixed while content is adapted without endangering the goal. Agile can be combined with required assurance/documentation; safety-critical context alone cannot prove it is unsuitable.

Study-note section SE05

> A user story states a small user need and value, supported by acceptance criteria; an epic groups broader work.

Question sources

Software Engineering review questions · Question 21

Question 250 | NS0766

The Spiral Model distinguishes itself from other models primarily through its emphasis on:

A. Sequential phase completion

B. Risk Analysis at every cycle

C. Customer delivery at the end of the project

D. Strict adherence to initial requirements

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer B: Risk analysis and reduction in every spiral cycle.

SE06 defines spiral development as risk driven, with each cycle identifying objectives and alternatives, assessing and reducing risks, producing a result and planning commitment. That emphasis distinguishes it from merely repeating phases or delivering only at the end. It can adapt requirements and combine other activities rather than enforcing an unchangeable initial specification.

Why each choice is right or wrong

A - Incorrect: A spiral can contain staged work, but sequential completion is not its defining driver.

B - Correct: Risk assessment and reduction guide each cycle’s work and further commitment.

C - Incorrect: Only delivering at the end is not the distinguishing spiral principle.

D - Incorrect: A risk-driven cycle can revise plans and requirements rather than rigidly freeze them.

Primary note proof

Study-note section SE06

> Spiral development is risk driven. Each cycle identifies objectives and alternatives, assesses and reduces important risks, develops/verifies an appropriate next result, and plans further commitment. A prototype may test a risky technology before a large investment. The amount and type of work depend on the risks.

Question sources

Software Engineering review questions · Question 22

Question 251 | NS0767

The V-Model derives its name from its shape. What does the BOTTOM of the 'V' represent?

A. Requirements Gathering

B. Deployment

C. Coding/Implementation

D. Maintenance

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Coding/implementation.

SE03 explicitly places coding at the bottom of the V while specification and design stages correspond to later testing levels. Requirements therefore begin the development side rather than occupy the bottom; deployment and maintenance are not that connecting implementation activity. This is a model-diagram convention, not a claim that all real work follows a perfect geometric sequence.

Why each choice is right or wrong

A - Incorrect: Requirements occur on the specification side, not at the coding bottom of the V.

B - Incorrect: Deployment is not the implementation activity joining the V-model’s sides.

C - Correct: The note explicitly identifies coding as the bottom of the V.

D - Incorrect: Maintenance occurs in lifecycle evolution rather than at this implementation point.

Primary note proof

Study-note section SE03

> The V-model pairs specification/design stages with later test levels. Requirements inform acceptance tests; system design informs system tests; component/detail design informs integration/unit tests. Coding lies at the bottom. Test planning starts early. Stable requirements and strong assurance may favor this organization, but neither a V shape nor waterfall guarantees safety or correctness.

Question sources

Software Engineering review questions · Question 23

Question 252 | NS0768

Which organization standardizes and maintains UML as a modeling language?

A. IEEE

B. ISO

C. Object Management Group (OMG)

D. W3C

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Object Management Group (OMG)

The note identifies the Object Management Group as UML's standardizing and maintaining organization. UML is a modeling language rather than a software lifecycle process. This stewardship question does not claim that the organization originally invented every contributing notation. IEEE, ISO and W3C are not the organization to which the supplied note attributes this particular standardization and maintenance role.

Why each choice is right or wrong

A - Incorrect: IEEE is not the attributed UML steward.

B - Incorrect: ISO is not the organization attributed this role in the note.

C - Correct: OMG is the stated standardizing and maintaining organization.

D - Incorrect: W3C is not the attributed UML steward.

Primary note proof

Study-note section SE17

> A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links. UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process. Structural things describe conceptual or physical structural elements.

Question sources

Software Engineering review questions · Question 24

Question 253 | NS0769

In a UML Use Case Diagram, the rectangle enclosing all use cases is called:

A. System Context

B. System Boundary Box

C. Package Container

D. Module Frame

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: The system boundary box.

SE14 says actors are outside the system-boundary rectangle and use-case ovals are inside. The enclosing rectangle therefore marks the modeled system's boundary. A context may be communicated by the diagram, but it is not the specific box name requested; packages and modules serve different grouping roles rather than defining this external actor boundary.

Why each choice is right or wrong

A - Incorrect: System context is broader information, not the specific name of this enclosing boundary rectangle.

B - Correct: The rectangle separates the modeled system's use cases from external actor roles.

C - Incorrect: A package groups modeling elements but is not the stated system/actor boundary.

D - Incorrect: Module frame is not the use-case system-boundary term defined in SE14.

Primary note proof

Study-note section SE14

> Draw actors outside the system boundary rectangle and use-case ovals inside. Name a goal with a verb and noun. The diagram shows roles, goals and relationships, not the full detailed algorithm or UI layout. Being an internal component is different from being an external actor; the chosen boundary determines which role applies.

Question sources

Software Engineering review questions · Question 25

Question 254 | NS0770

When execution reaches an include relationship's inclusion point in a base use case, what does include require?

A. Optional behavior that may be added to the base use case

B. The included behavior is required whenever execution reaches the inclusion point.

C. An inheritance relationship between two use cases

D. A timing constraint between two use cases

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: The included behavior is required whenever execution reaches the inclusion point.

Include factors required reused behavior into another use case. When the base reaches the inclusion point, the included behavior is incorporated and performed as specified. This does not imply that every possible conditional branch of the base reaches that point. Optional conditional additions correspond to extend, specialization to generalization, and timing constraints are separate from the inclusion relationship.

Why each choice is right or wrong

A - Incorrect: Conditional optional added behavior describes extend.

B - Correct: The included behavior is required when its inclusion point is reached.

C - Incorrect: Specialization is generalization rather than include.

D - Incorrect: Include is not a timing constraint.

Primary note proof

Study-note section SE16

> Generalization relates a specialized actor/use case to a more general one, with a hollow triangle toward the general element. An abstract use case specifies behavior for specialization and is not directly instantiated as a complete use case. It is not defined merely by reuse or lack of a direct actor line. Included behavior is incorporated where the inclusion is reached; do not infer every conditional branch in the base always executes.

Study-note section SE16

> Include factors required reused behavior: Submit Order includes Validate Order, with the dashed arrow from the including/base use case to the included one.

Question sources

Software Engineering review questions · Question 26

Question 255 | NS0771

In a UML class diagram, visibility marked with '#' indicates:

A. Public access

B. Private access

C. Protected access

D. Package-level access

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Protected visibility.

SE17 lists + public, - private, # protected and ~ package. The requested hash marker therefore means protected access. These symbols indicate visibility of a class member; they do not express object multiplicity, inheritance direction or class ownership, and the other choices correspond to different explicit markers.

Why each choice is right or wrong

A - Incorrect: Public visibility is marked with + rather than #.

B - Incorrect: Private visibility is marked with - rather than #.

C - Correct: The note explicitly assigns # to protected visibility.

D - Incorrect: Package visibility uses ~ rather than the protected # marker.

Primary note proof

Study-note section SE17

> A class box lists name, attributes and operations; + public, - private, # protected and ~ package indicate visibility. A class diagram shows static types and relationships, while an object diagram shows particular instances/links. UML is maintained and standardized by OMG; it is a modeling language, not a lifecycle process. Structural things describe conceptual or physical structural elements.

Question sources

Software Engineering review questions · Question 27

Question 256 | NS0772

In UML, which relationship is represented by a HOLLOW diamond shape pointing toward the 'whole' class?

A. Composition

B. Dependency

C. Aggregation

D. Generalization

Hint

Match the information to be represented with each model’s purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Shared aggregation.

SE18 defines shared aggregation with a hollow diamond at the whole end. Composition instead uses a filled diamond; dependency is a dashed arrow and generalization uses a hollow triangle. The question’s whole-end description is consistent with the notation, so the hollow shape uniquely identifies aggregation among these alternatives.

Why each choice is right or wrong

A - Incorrect: Composition uses a filled diamond, not the specified hollow diamond.

B - Incorrect: Dependency is represented by a dashed arrow rather than a whole-part diamond.

C - Correct: Shared aggregation uses a hollow diamond placed at the whole end.

D - Incorrect: Generalization uses a hollow triangle toward the general element rather than a diamond.

Primary note proof

Study-note section SE18

> Composition uses a filled diamond at the whole end and strong ownership: a part is in at most one composite at a time, and parts still owned are destroyed with that composite. Parts may be detached/transferred under a defined model. Thus “cannot ever exist independently” is an oversimplification. For Car composing Engine, the diamond belongs beside Car; saying it points to Engine is imprecise. The Car-Engine example describes composition.

Study-note section SE18

> Shared aggregation uses a hollow diamond at the whole end and weak whole-part meaning;

Question sources

Software Engineering review questions · Question 28

Question 257 | NS0773

Which statement accurately distinguishes UML composition from shared aggregation while allowing model-defined detachment or transfer?

A. Aggregation is for classes; Composition is for interfaces

B. Composition has exclusive composite ownership at a time and destroys still-owned parts with the whole; shared aggregation gives weaker whole-part meaning.

C. Aggregation implies inheritance; Composition does not

D. They are synonymous in modern UML

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: Composition has exclusive composite ownership at a time and destroys still-owned parts with the whole; shared aggregation gives weaker whole-part meaning.

Composition gives strong ownership: a part is in at most one composite at a time, and still-owned parts are destroyed with that composite. Shared aggregation indicates weaker whole-part meaning without that detailed lifecycle policy. A model can permit detachment or transfer, so permanent inability to exist independently is too strong. Neither relationship is inheritance, and composition is not restricted to interfaces.

Why each choice is right or wrong

A - Incorrect: Composition is not restricted to interfaces.

B - Correct: This states strong ownership and destruction of still-owned parts, unlike weak shared aggregation.

C - Incorrect: Aggregation is not inheritance; generalization is separate.

D - Incorrect: The ownership meanings differ, so they are not synonyms.

Primary note proof

Study-note section SE18

> Composition uses a filled diamond at the whole end and strong ownership: a part is in at most one composite at a time, and parts still owned are destroyed with that composite. Parts may be detached/transferred under a defined model. Thus "cannot ever exist independently" is an oversimplification. For Car composing Engine, the diamond belongs beside Car; saying it points to Engine is imprecise. The Car-Engine example describes composition.

Question sources

Software Engineering review questions · Question 29

Question 258 | NS0774

Under the course's sequence-diagram layout heuristic-rather than a mandatory UML column rule-which participant belongs in the second column?

A. The control object managing the use case

B. The actor who initiated the use case

C. A boundary object the actor used to start the use case

D. An entity object central to the use case

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: A boundary object the actor used to start the use case

The course's heuristic places the initiating actor first, the boundary object second, the control object third and entity objects afterward. The boundary used by the actor therefore occupies the second column under that convention. UML itself does not mandate this column order; lifeline semantics come from the modeled participants and messages rather than their horizontal position alone.

Why each choice is right or wrong

A - Incorrect: The heuristic places control third.

B - Incorrect: The initiating actor is first.

C - Correct: The actor-facing boundary is second under the specified heuristic.

D - Incorrect: Entities follow the boundary and control in this heuristic.

Primary note proof

Study-note section SE19

> One interaction-diagram layout heuristic puts the initiating actor first, boundary second, control third, then entities. This is not a required UML column order. In ReportEmergency, FieldOfficer activates ReportEmergencyButton, enters details through EmergencyReportForm, and later sees AcknowledgmentNotice; DispatcherStation belongs to the dispatcher side. Choose the button for initiating this particular scenario.

Question sources

Software Engineering review questions · Question 30

Question 259 | NS0775

In a UML Activity Diagram, a diamond shape is used to represent:

A. Initial State

B. Final State

C. Action Box

D. Decision Box

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer D: A decision node, also usable as a merge node in the appropriate flow.

SE20 names the guarded decision diamond and separately notes that the same shape can merge alternative flows. D is therefore a correct use of the shape, without claiming decision is its only role. Initial nodes are filled circles, actions are rounded, and final-state notation is not the diamond described.

Why each choice is right or wrong

A - Incorrect: The initial node is a filled circle rather than a diamond.

B - Incorrect: The diamond is not the standard final-state marker in the described notation.

C - Incorrect: Actions are rounded nodes rather than guarded-flow diamonds.

D - Correct: A diamond can represent a guarded decision; it can also merge alternative flows.

Primary note proof

Study-note section SE20

> An activity diagram models workflow: filled circle start, rounded actions, guarded decision diamond, thick fork/join bars and partitions/swimlanes for responsibility. A diamond can also merge alternative flows; a join synchronizes concurrent flows. A state machine models states and transitions triggered by events, often with guards/actions, for one object's lifecycle.

Question sources

Software Engineering review questions · Question 31

Question 260 | NS0776

In system design, 'coupling' between subsystems refers to:

A. How well objects within a subsystem work toward a common goal

B. The number of operations a subsystem exposes

C. The degree of dependency between two subsystems

D. The architectural pattern chosen for the system

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: The degree of dependency between subsystems.

SE22 explicitly distinguishes cohesion within a module from coupling between modules. A describes cohesion; C describes coupling. Interface operation count can affect design complexity but is not the definition of dependence, and an architectural pattern may influence coupling without being the relationship measure itself. The goal is usually low coupling with high cohesion.

Why each choice is right or wrong

A - Incorrect: Related purpose inside a subsystem defines cohesion rather than between-subsystem coupling.

B - Incorrect: Operation count alone does not measure the actual dependencies between subsystems.

C - Correct: Coupling is the degree of dependency between modules or subsystems.

D - Incorrect: An architectural style can affect coupling but is not its definition.

Primary note proof

Study-note section SE22

> Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module’s internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Question sources

Software Engineering review questions · Question 32

Question 261 | NS0777

In architectural design, a 'closed architecture' in a layered system means:

A. No external access to any layer is allowed

B. A layer can access any layer at deeper levels freely

C. A layer can only call the layer immediately below it

D. All layers communicate through a central repository

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: Calls are restricted to the immediately lower layer.

SE23 defines a closed layered architecture as allowing calls only to the immediately lower layer. Open layering may permit approved skipping, so B describes a different policy. Closed does not mean no external access or a shared repository; it concerns permitted dependencies between layers rather than the system's exposure or storage organization.

Why each choice is right or wrong

A - Incorrect: Closed layering restricts interlayer calls, not every form of external access.

B - Incorrect: Skipping freely to deeper layers contradicts the immediate-lower-layer restriction.

C - Correct: A layer calls only the immediately lower layer in the defined closed architecture.

D - Incorrect: A central repository is another architectural arrangement and does not define closed layering.

Primary note proof

Study-note section SE23

> A layer is a logical abstraction level; a tier is a deployment separation. A closed layered architecture permits calls only to the immediately lower layer; an open one permits approved skipping. For top-to-bottom A,B,C,D, A→B obeys closed layering, A→C skips, and lower-to-upper calls violate the simple dependency rule unless explicitly mediated by a defined mechanism.

Question sources

Software Engineering review questions · Question 33

Question 262 | NS0778

Which architectural style divides subsystems into Model, View, and Controller components?

A. Repository Style

B. Client/Server Style

C. Peer-to-Peer Style

D. Model-View-Controller Style

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer D: Model-View-Controller.

SE24 explicitly defines MVC as separating a domain model, display view and interaction controller. D names that exact style. Repository, client-server and peer-to-peer organize central data or interacting roles differently; any may be combined with MVC, but none is the name of this particular three-responsibility separation.

Why each choice is right or wrong

A - Incorrect: Repository style centers shared data rather than the specific model/view/controller separation.

B - Incorrect: Client-server separates requesters and service providers, not these three presentation-interaction responsibilities.

C - Incorrect: Peer-to-peer allows participants to play requester and provider roles rather than defining MVC components.

D - Correct: Model-View-Controller explicitly names the three responsibilities in the question.

Primary note proof

Study-note section SE24

> Repository style shares central data, simplifying common access but creating bottleneck/schema dependencies. MVC separates domain model, display view and interaction controller. Client-server separates requesters from service providers; peer-to-peer permits both roles. Pipe-and-filter applies successive transformations to data streams. Three-tier separates presentation, application logic and storage; some four-tier variants split presentation client/server.

Question sources

Software Engineering review questions · Question 34

Question 263 | NS0779

In the Three-Tier architectural style, which layer handles 'storing, retrieving, and querying persistent data'?

A. Interface Layer

B. Application Logic Layer

C. Storage Layer

D. Presentation Layer

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer C: The storage layer/tier.

SE24 separates three-tier presentation, application logic and storage. Persistent data storage and retrieval belong to the storage responsibility, whereas presentation handles interaction and application logic coordinates domain behavior. The question uses layer for a tier-style role; that terminology does not change which responsibility owns persistence.

Why each choice is right or wrong

A - Incorrect: The interface responsibility handles interaction rather than persistent storage and querying.

B - Incorrect: Application logic coordinates domain operations but delegates persistence to the storage responsibility.

C - Correct: The storage role handles persistent data, including retrieval and queries.

D - Incorrect: Presentation formats user interaction and output rather than owning persistent data storage.

Primary note proof

Study-note section SE24

> Repository style shares central data, simplifying common access but creating bottleneck/schema dependencies. MVC separates domain model, display view and interaction controller. Client-server separates requesters from service providers; peer-to-peer permits both roles. Pipe-and-filter applies successive transformations to data streams. Three-tier separates presentation, application logic and storage; some four-tier variants split presentation client/server.

Study-note section SE23

> A layer is a logical abstraction level; a tier is a deployment separation. A closed layered architecture permits calls only to the immediately lower layer; an open one permits approved skipping. For top-to-bottom A,B,C,D, A→B obeys closed layering, A→C skips, and lower-to-upper calls violate the simple dependency rule unless explicitly mediated by a defined mechanism.

Question sources

Software Engineering review questions · Question 35

Question 264 | NS0780

In the analysis model, which type of object represents the bridges between the system and the actors?

A. Entity Objects

B. Control Objects

C. Boundary Objects

D. Interface Objects

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Boundary objects.

SE13 defines boundary objects as mediating interaction, with ReportForm as an example. That is the bridge between external actors and the system. Entity objects hold domain information, while controls coordinate use cases. Interface objects is a generic descriptive phrase, but boundary is the explicitly defined analysis-category name requested among these choices.

Why each choice is right or wrong

A - Incorrect: Entity objects represent domain information rather than the actor-facing interaction bridge.

B - Incorrect: Control objects coordinate use-case behavior rather than primarily mediate actor interaction.

C - Correct: Boundary objects are explicitly defined as mediating system-actor interaction.

D - Incorrect: Interface objects is not the named entity/boundary/control category specified here's analysis model.

Primary note proof

Study-note section SE13

> A scenario is one concrete execution; a use case abstracts the related successful and alternative scenarios. Analysis integrates functional (use cases/scenarios), structural/object (classes) and dynamic (states/interactions) models. Entity objects represent domain information such as Incident; boundary objects mediate interaction, such as ReportForm; control objects coordinate a use case. A UI form is normally boundary, not entity.

Question sources

Software Engineering review questions · Question 36

Question 265 | NS0781

Which heuristic rule states that there should be 'one control object per actor' in a use case?

A. CRC Card Heuristic

B. Control Object Identification Heuristic

C. Boundary Object Heuristic

D. Sequence Diagram Column Rule

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: The control-object identification heuristic.

SE13 permits one control per use case or participating actor as a useful starting point, while denying a mandatory UML rule. The stem already asks for a heuristic, so B correctly identifies the control-object selection guideline. CRC, boundary identification and column layout concern different tasks; none makes the count a required lifetime or architecture constraint.

Why each choice is right or wrong

A - Incorrect: CRC records class responsibilities and collaborators rather than prescribing this candidate-control count.

B - Correct: This is the heuristic for identifying candidate use-case control objects, not a universal UML constraint.

C - Incorrect: Boundary heuristics identify interaction-facing objects rather than coordinating controls.

D - Incorrect: Sequence-column layout orders participants visually and does not define candidate control-object counts.

Primary note proof

Study-note section SE13

> Use nouns as candidate objects, possessive descriptions as candidate attributes, and verbs such as manages/reports-to as candidate associations. These are heuristics, not automatic parsing rules. CRC means Class, Responsibilities, Collaborators. A control per use case or participating actor can be a useful starting point, not a mandatory UML rule or fixed object lifetime.

Question sources

Software Engineering review questions · Question 37

Question 266 | NS0782

CRC cards stand for:

A. Class, Requirements, Conditions

B. Component, Relationship, Constraints

C. Class, Responsibilities, Collaborators

D. Control, Reuse, Composition

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: Class, Responsibilities, Collaborators.

SE13 expands CRC directly as Class, Responsibilities, Collaborators. The card identifies the class, what it must do or know, and which other objects help fulfill those responsibilities. The distractors replace these terms with requirements, components or unrelated design concepts and therefore do not preserve the stated expansion.

Why each choice is right or wrong

A - Incorrect: Requirements and Conditions are not the R and final C in the stated expansion.

B - Incorrect: Component, Relationship and Constraints do not match the class-card terminology.

C - Correct: This is the exact CRC expansion supplied in SE13.

D - Incorrect: Control, Reuse and Composition are different concepts rather than the CRC card fields.

Primary note proof

Study-note section SE13

> Use nouns as candidate objects, possessive descriptions as candidate attributes, and verbs such as manages/reports-to as candidate associations. These are heuristics, not automatic parsing rules. CRC means Class, Responsibilities, Collaborators. A control per use case or participating actor can be a useful starting point, not a mandatory UML rule or fixed object lifetime.

Question sources

Software Engineering review questions · Question 38

Question 267 | NS0783

In UML association multiplicity, what does '0..*' mean?

A. Exactly one instance

B. Zero or exactly one instance

C. Zero or more instances

D. One or more instances

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Zero or more instances.

SE17 defines 0..* as any nonnegative number. Zero is allowed and the star places no finite upper bound, so C fits. The alternatives correspond to 1, 0..1 and 1..* respectively. Multiplicity at one end is interpreted relative to one object at the opposite end, not as a whole-system object count.

Why each choice is right or wrong

A - Incorrect: Exactly one corresponds to multiplicity 1 rather than 0..*.

B - Incorrect: Optional single-valued multiplicity is 0..1, not an unbounded star.

C - Correct: Zero is permitted and the upper star allows any number of related instances.

D - Incorrect: One or more requires lower bound 1, written 1..*.

Primary note proof

Study-note section SE17

> Multiplicity at one end counts how many objects there may correspond to one object at the opposite end: 1 exactly one, 0..1 optional, 0..* any nonnegative number, 1..* at least one. A one-to-many association does not automatically mean composition. Read each end separately before choosing cardinality.

Question sources

Software Engineering review questions · Question 39

Question 268 | NS0784

In Abbott's heuristics for identifying associations, which phrase pattern most directly suggests a relationship between classes?

A. Look for adjective phrases describing class properties

B. Examine verb phrases like 'has', 'is part of', 'manages', 'reports to'

C. Find all nouns that appear more than twice

D. Identify all user interface terms

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer B: Verb phrases expressing relationships, such as manages or reports to.

SE13 recommends nouns as candidate objects and verbs such as manages/reports-to as candidate associations. B is the relationship-phrase option. Adjectives generally describe properties, repeated nouns do not automatically imply an association, and UI terms suggest boundary concerns. These are candidate-identification heuristics rather than a proof that every verb must become a modeled link.

Why each choice is right or wrong

A - Incorrect: Property descriptions suggest attributes rather than directly identifying links between classes.

B - Correct: Relationship verbs are the note’s direct heuristic for candidate associations.

C - Incorrect: Noun frequency is not the stated association-identification rule; nouns instead suggest candidate objects.

D - Incorrect: UI terminology may suggest boundaries but does not directly establish an association between domain classes.

Primary note proof

Study-note section SE13

> Use nouns as candidate objects, possessive descriptions as candidate attributes, and verbs such as manages/reports-to as candidate associations. These are heuristics, not automatic parsing rules. CRC means Class, Responsibilities, Collaborators. A control per use case or participating actor can be a useful starting point, not a mandatory UML rule or fixed object lifetime.

Question sources

Software Engineering review questions · Question 40

Question 269 | NS0785

In software testing, which type of testing evaluates the software's behavior WITHOUT knowledge of its internal code structure?

A. White Box Testing

B. Grey Box Testing

C. Black Box Testing

D. Structural Testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: Black-box testing.

SE36 contrasts black-box cases derived from required behavior with white-box cases derived from implementation structure. Grey-box uses partial internal knowledge. The question excludes internal-code knowledge, so black-box is the appropriate technique. Structural testing instead refers to the very internal paths or statements that this scenario does not inspect.

Why each choice is right or wrong

A - Incorrect: White-box testing explicitly uses internal implementation structure to derive cases.

B - Incorrect: Grey-box testing uses partial internal knowledge rather than excluding it.

C - Correct: Black-box testing derives cases from externally specified behavior without needing code structure.

D - Incorrect: Structural testing is based on internal control or code structure, contrary to the stem.

Primary note proof

Study-note section SE36

> Black-box techniques derive cases from specified behavior; white-box techniques derive them from internal structure; grey-box uses partial internal knowledge. Regression testing checks for unintended effects after change; confirmation testing checks the particular defect repair. Exploratory testing combines learning, design and execution and is experience based. Manual/automated describe execution support, not a compulsory black-/white-box classification.

Question sources

Software Engineering review questions · Question 41

Question 270 | NS0786

Which type of functional testing verifies that individual components work correctly in ISOLATION?

A. System Testing

B. Integration Testing

C. Regression Testing

D. Unit Testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer D: Unit testing.

SE35 defines unit testing as isolating a function, class or small component, often replacing dependencies with stubs or mocks. Integration checks combined interactions and system testing evaluates the whole system. Regression describes testing after changes and can occur at different levels, so it is not the isolation level named by the question.

Why each choice is right or wrong

A - Incorrect: System testing evaluates the complete integrated system rather than an isolated component.

B - Incorrect: Integration testing focuses on interactions among combined parts, not isolation.

C - Incorrect: Regression checks unintended effects after change and is not inherently a component-isolation level.

D - Correct: Unit testing evaluates a small component in isolation, with dependencies replaceable by test doubles.

Primary note proof

Study-note section SE35

> Unit testing isolates a function, class or small component, often replacing dependencies with stubs/mocks; a driver invokes the unit and a stub substitutes a called dependency. Integration testing exercises interfaces and interactions among combined parts. System testing checks the complete integrated system against functional and nonfunctional requirements in a representative environment.

Question sources

Software Engineering review questions · Question 42

Question 271 | NS0787

Non-functional testing that evaluates how the system behaves under expected workloads and measures response speed is called:

A. Usability Testing

B. Reliability Testing

C. Performance Testing

D. Compatibility Testing

Hint

Separate testing scope, technique and purpose. Use the stated workload, observable result or coverage target rather than a universal ranking.

Answer C: Performance testing, specifically load-oriented testing under expected demand.

SE36 defines performance tests as assessing speed, throughput and resource use under a specified workload and distinguishes expected load from stress beyond normal limits. The stem explicitly adds workload and response speed, so it supports performance testing rather than merely a vague normal-condition test. Usability, reliability and compatibility target different quality questions.

Why each choice is right or wrong

A - Incorrect: Usability focuses on task success and ease of use rather than response-speed measurements under load.

B - Incorrect: Reliability concerns failure-free service, not primarily the stated response-time criterion.

C - Correct: Speed under an expected workload is performance/load testing as defined in SE36.

D - Incorrect: Compatibility examines cooperating environments rather than the specified speed-under-load behavior.

Primary note proof

Study-note section SE36

> Functional tests check required behavior against the SRS and acceptance criteria. Performance tests assess speed, throughput and resources under a specified workload; load uses expected demands and stress exceeds normal limits. Usability, security, compatibility and reliability tests target their respective qualities. “Normal operating conditions” alone does not distinguish functional from performance testing.

Question sources

Software Engineering review questions · Question 43

Question 272 | NS0788

In the analysis phase, which model type uses use cases and scenarios to describe what the software should do?

A. Dynamic Model

B. Analysis Object Model

C. Functional Model

D. Structural Model

Hint

Distinguish observable services, measurable qualities and implementation constraints. Check scope, stakeholder meaning and the requested level of specificity.

Answer C: The functional model.

SE13 explicitly associates use cases and scenarios with functional modeling, classes with structural/object modeling, and states/interactions with dynamic modeling. Thus C names the requested description of what the system should do. The alternatives describe state-dependent behavior or static object structure rather than this user-goal and scenario view.

Why each choice is right or wrong

A - Incorrect: Dynamic models emphasize states and interactions rather than the named use-case/scenario category.

B - Incorrect: The analysis object model captures classes and relationships rather than the functional use-case view.

C - Correct: Use cases and scenarios are explicitly the functional analysis model.

D - Incorrect: Structural modeling describes static classes and links rather than required user-goal flows.

Primary note proof

Study-note section SE13

> A scenario is one concrete execution; a use case abstracts the related successful and alternative scenarios. Analysis integrates functional (use cases/scenarios), structural/object (classes) and dynamic (states/interactions) models. Entity objects represent domain information such as Incident; boundary objects mediate interaction, such as ReportForm; control objects coordinate a use case. A UI form is normally boundary, not entity.

Question sources

Software Engineering review questions · Question 44

Question 273 | NS0789

In an analysis model, an object coordinates a use case and moves among Idle, WaitingForReply and Completing states as events arrive. Domain entities are passive records and the form has no persistent modes in this scenario. Which object category is the direct state-machine target described?

A. Entity Objects

B. Boundary Objects

C. Control Objects

D. All objects equally

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Control Objects

A control object coordinates a use case, and the described coordinator has significant event-dependent states. A state machine is therefore directly useful for that control object in this scenario. Entities and boundaries can also have meaningful lifecycles in other systems, but here their stated behavior does not supply the modeled states. Suitability follows from responsibilities and lifecycle behavior, not equal need across all objects.

Why each choice is right or wrong

A - Incorrect: The scenario's entities are passive records rather than the stated coordinator.

B - Incorrect: The form has no persistent modes and is not the use-case coordinator.

C - Correct: The stateful coordinator is a control object.

D - Incorrect: The need depends on significant state behavior, not equality across every object.

Primary note proof

Study-note section SE20

> Choose a sequence diagram for message order between participants, a state machine for event-dependent state changes, and activity for workflow. Both sequence and state diagrams describe behavior over time. State machines are useful for objects with significant states, whether entity or control. A DFD shows data stores/flows and transformations from inputs to outputs; it is not a UML diagram.

Study-note section SE13

> A scenario is one concrete execution; a use case abstracts the related successful and alternative scenarios. Analysis integrates functional (use cases/scenarios), structural/object (classes) and dynamic (states/interactions) models. Entity objects represent domain information such as Incident; boundary objects mediate interaction, such as ReportForm; control objects coordinate a use case. A UI form is normally boundary, not entity.

Question sources

Software Engineering review questions · Question 45

Question 274 | NS0790

In the ReportEmergency use case example, which boundary object allows the FieldOfficer to initiate the reporting process?

A. EmergencyReportForm

B. AcknowledgmentNotice

C. ReportEmergencyButton

D. DispatcherStation

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: ReportEmergencyButton.

SE19 explicitly says FieldOfficer activates ReportEmergencyButton, supplies details through EmergencyReportForm and later sees AcknowledgmentNotice. DispatcherStation belongs to the dispatcher side. Therefore the button initiates this particular scenario; the form handles later input and the notice reports completion. This is grounded in the named example rather than a generic rule that every use case starts with a button.

Why each choice is right or wrong

A - Incorrect: The form receives incident details after initiation rather than starting the documented scenario.

B - Incorrect: The acknowledgment is seen later as feedback, not used to initiate reporting.

C - Correct: The note explicitly identifies this button as the FieldOfficer's initiating boundary object.

D - Incorrect: DispatcherStation is on the dispatcher side rather than the FieldOfficer's initiation interface.

Primary note proof

Study-note section SE19

> One interaction-diagram layout heuristic puts the initiating actor first, boundary second, control third, then entities. This is not a required UML column order. In ReportEmergency, FieldOfficer activates ReportEmergencyButton, enters details through EmergencyReportForm, and later sees AcknowledgmentNotice; DispatcherStation belongs to the dispatcher side. Choose the button for initiating this particular scenario.

Question sources

Software Engineering review questions · Question 46

Question 275 | NS0791

Generalization in UML modeling is best described as:

A. Dividing a class into multiple subclasses with no shared properties

B. Identifying a general concept where other concepts are more specific versions of it

C. Creating interfaces for external system communication

D. Grouping unrelated classes into a package for organization

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer B: A general concept with more-specific specializations.

SE16 relates a specialized actor or use case to a more general one; SE18 gives the corresponding class generalization toward a superclass. B expresses that abstraction/specialization relationship. It does not merely package unrelated classes, create an external interface or split classes while denying shared general properties.

Why each choice is right or wrong

A - Incorrect: Specialization is related through a general concept rather than requiring no shared properties.

B - Correct: Generalization identifies the common general element of which other concepts are more specific versions.

C - Incorrect: External communication interfaces do not by themselves define a generalization relationship.

D - Incorrect: Packaging unrelated classes is organizational grouping rather than specialization of a shared general concept.

Primary note proof

Study-note section SE18

> Association connects related instances. A dashed dependency arrow means a client uses a supplier. Generalization uses a solid line and hollow triangle pointing to the superclass/general element. Shared aggregation uses a hollow diamond at the whole end and weak whole-part meaning; it does not by itself define a detailed lifecycle policy.

Study-note section SE16

> Generalization relates a specialized actor/use case to a more general one, with a hollow triangle toward the general element. An abstract use case specifies behavior for specialization and is not directly instantiated as a complete use case. It is not defined merely by reuse or lack of a direct actor line. Included behavior is incorporated where the inclusion is reached; do not infer every conditional branch in the base always executes.

Question sources

Software Engineering review questions · Question 47

Question 276 | NS0792

The 'high cohesion, low coupling' principle in subsystem design means:

A. Subsystems should be tightly interconnected for maximum communication efficiency

B. Objects within a subsystem should be closely related, while subsystems themselves should be mostly independent

C. Each subsystem should have minimal internal functionality to reduce complexity

D. All subsystems should share a single central data repository

Hint

Separate responsibilities inside a component from dependencies across boundaries. Evaluate the stated structure, change and reuse goals without assuming guaranteed benefits.

Answer B: Related responsibilities within a subsystem, limited dependencies between subsystems.

SE22 defines cohesion as related purpose within a module and coupling as dependency across modules, then recommends high cohesion and low coupling. B states both dimensions. It does not require minimal functionality, tight cross-connections or one shared repository; those choices can undermine coherent responsibilities or increase dependency rather than establish the principle.

Why each choice is right or wrong

A - Incorrect: Tight interconnection increases coupling rather than meeting the low-coupling goal.

B - Correct: Closely related internal responsibilities and limited external dependencies express both parts of the principle.

C - Incorrect: Cohesion concerns related purpose, not simply having as little functionality as possible.

D - Incorrect: A shared repository is a style choice and can create dependencies; it does not define cohesion or coupling.

Primary note proof

Study-note section SE22

> Cohesion measures how well elements inside a module serve a related purpose; coupling measures dependency between modules. Seek high cohesion and low coupling. Shared mutable data and knowledge of another module’s internal representation increase coupling. Information hiding exposes stable operations while concealing changeable decisions.

Question sources

Software Engineering review questions · Question 48

Question 277 | NS0793

A safety-critical contract requires traceable assurance records and reviewed verification before release. Waterfall, V-model and spiral proposals include those obligations; the proposed lightweight Agile variant supplies working code but omits the required assurance records. Which proposal is least appropriate as specified?

A. Waterfall Model

B. V-Model

C. Agile Model

D. Spiral Model

Hint

Identify whether the scenario prioritizes delivery, learning, adaptation or risk reduction. Several processes can share individual practices, so compare their defining emphasis.

Answer C: Agile Model

The specified Agile proposal omits mandatory assurance records, so it cannot satisfy the stated contract without modification. The other proposals include those obligations and are not excluded on that ground. This is a judgment about the supplied lightweight variant, not Agile in general: the note explicitly allows Agile to incorporate required documentation and assurance. Stable requirements alone would not establish a universal ranking.

Why each choice is right or wrong

A - Incorrect: This proposal includes the required reviewed deliverables.

B - Incorrect: This proposal includes the required assurance obligations.

C - Correct: The specified lightweight variant omits mandatory records and is therefore least suitable as proposed.

D - Incorrect: This proposal includes the stated assurance obligations.

Primary note proof

Study-note section SE05

> Scrum uses timeboxed Sprints of at most one month, with a Sprint Goal, selected work and a usable Increment meeting a Definition of Done. Scope is discussed before and during planning; it is not forbidden to agree scope. Timeboxes tend to stay fixed while content is adapted without endangering the goal. Agile can be combined with required assurance/documentation; safety-critical context alone cannot prove it is unsuitable.

Study-note section SE03

> The V-model pairs specification/design stages with later test levels. Requirements inform acceptance tests; system design informs system tests; component/detail design informs integration/unit tests. Coding lies at the bottom. Test planning starts early. Stable requirements

and strong assurance may favor this organization, but neither a V shape nor waterfall guarantees safety or correctness.

Study-note section SE03

Classical waterfall separates planned phases with reviewed deliverables. It fits well-understood, stable requirements and familiar technology; late feedback makes changes costly.

Question sources

Software Engineering review questions · Question 49

Question 278 | NS0794

In a sequence diagram, what does an 'X' (cross) at the end of a lifeline dashed line indicate?

A. The object is temporarily inactive

B. The object is waiting for a message

C. The object ceases to exist at that point

D. The object is calling itself recursively

Hint

Match the information to be represented with each model's purpose. Read relationship direction, boundary and ownership conditions before classifying it.

Answer C: Modeled destruction of the object at that point.

SE19 explicitly identifies X as modeled destruction, so C is correct in UML diagram semantics. Temporary inactivity and waiting do not end an object's modeled existence, and a recursive message is represented by an interaction rather than destruction. The marker does not guarantee the runtime moment at which a garbage collector reclaims memory.

Why each choice is right or wrong

A - Incorrect: Temporary inactivity does not terminate the object's lifeline.

B - Incorrect: Waiting for a message does not mean the modeled object has been destroyed.

C - Correct: X marks the object's modeled destruction at that point, not necessarily immediate memory reclamation.

D - Incorrect: A recursive call is an interaction message and does not use the destruction marker.

Primary note proof

Study-note section SE19

> Sequence diagrams show interactions across lifelines, with order/time generally downward. Messages cross between participants; activation rectangles show execution. A creation message starts a lifeline; an X marks modeled destruction, not a guarantee about when a garbage collector runs. Synchronous calls wait for completion; asynchronous messages permit the sender to proceed.

Question sources

Software Engineering review questions · Question 50