CHAPTER 1 FUNDAMENTALS OF TESTING

What is Testing?

1- Identify typical objectives of testing (K1)

Typical Objectives of Testing

For any given project, the objectives of testing may include:

▶ To evaluate work products such as requirements, user stories, design, and code

▶ To verify whether all specified requirements have been fulfilled

▶ To validate whether the test object is complete and works as the users and other stakeholders expect

▶ To build confidence in the level of quality of the test object

▶ To prevent defects

▶ To find failures and defects

▶ To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object

▶ To reduce the level of risk of inadequate software quality (e.g., previously undetected failures occurring in operation)

▶ To comply with contractual, legal, or regulatory requirements or standards, and/or to verify the test object's compliance with such requirements or standards

The objectives of testing can vary, depending upon the context of the component or system being tested, the test level, and the software development lifecycle model. These differences may include, for example:

▶ During component testing, one objective may be to find as many failures as possible so that the underlying defects are identified and fixed early. Another objective may be to increase code coverage of the component tests.

▶ During acceptance testing, one objective may be to confirm that the system works as expected and satisfies requirements. Another objective of this testing may be to give information to stakeholders about the risk of releasing the system at a given time.

(Identify is more evidence based wording)

Question #2 (1 Point)

Which of the following statements is a valid objective for testing?

a) To determine whether enough component tests were executed within system testing.
b) To find as many failures as possible so that defects can be identified and corrected.
c) To prove that all possible defects are identified.
d) To prove that any remaining defects will not cause any failures.

Select one option.

2- Differentiate Testing from Debugging (K2)

Testing and debugging are different. Executing tests can show failures that are caused by defects in the software. Debugging is the development activity that finds, analyzes, and fixes such defects. Subsequent confirmation testing checks whether the fixes resolved the defects. In some cases, testers are responsible for the initial test and the final confirmation test, while developers do the debugging and associated component testing. However, in Agile development and in some other lifecycles, testers may be involved in debugging and component testing.

Question #3 (1 Point)

Which of the following statements correctly describes the difference between testing and debugging?

a) Testing identifies the source of defects; debugging analyzes the defects and proposes prevention activities.
b) Testing shows failures caused by defects; debugging finds, analyzes, and removes the causes of failures in the software.
c) Testing removes faults; debugging identifies the causes of failures.
d) Testing prevents the causes of failures; debugging removes the failures.

Select one option.

Why is Testing Necessary?

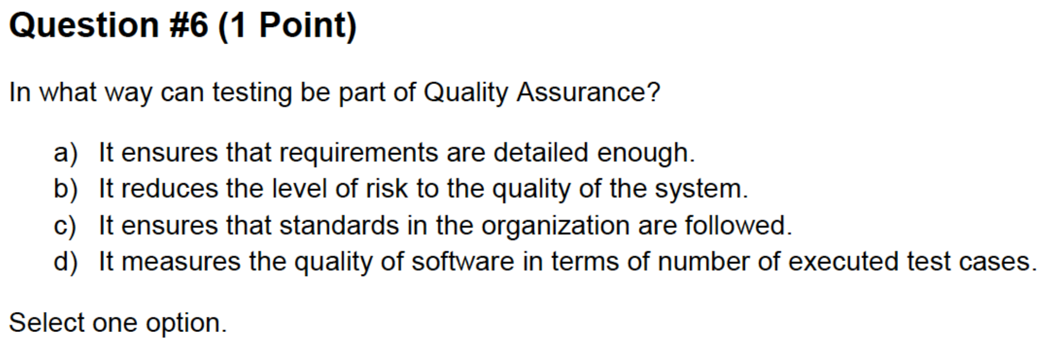
3- Give examples of why testing is necessary (K2)

Rigorous testing of components and systems, and their associated documentation, can help reduce the risk of failures occurring during operation. When defects are detected, and subsequently fixed, this contributes to the quality of the components or systems. In addition, software testing may also be required to meet contractual or legal requirements or industry-specific standards.

4- Describe the relationships between testing and quality assurance and give examples of how testing contributes to higher quality (K2)

Quality Assurance and Testing

While people often use the phrase quality assurance (or just QA) to refer to testing, quality assurance and testing are not the same, but they are related. A larger concept, quality management, ties them together. Quality management includes all activities that direct and control an organization with regard to quality. Among other activities, quality management includes both quality assurance and quality control. Quality assurance is typically focused on adherence to proper processes, in order to provide confidence that the appropriate levels of quality will be achieved. When processes are carried out properly, the work products created by those processes are generally of higher quality, which contributes to defect prevention. In addition, the use of root cause analysis to detect and remove the causes of defects, along with the proper application of the findings of retrospective meetings to improve processes, are important for effective quality assurance.



Question #6 (1 Point)

In what way can testing be part of Quality Assurance?

a) It ensures that requirements are detailed enough.
b) It reduces the level of risk to the quality of the system.
c) It ensures that standards in the organization are followed.
d) It measures the quality of software in terms of number of executed test cases.

Select one option.

Quality control involves various activities, including test activities, that support the achievement of appropriate levels of quality. Test activities are part of the overall software development or maintenance process. Since quality assurance is concerned with the proper execution of the entire process, quality assurance supports proper testing. As described in sections 1.1.1 and 1.2.1, testing contributes to the achievement of quality in a variety of ways.

(Rephrased to focus on relationships between testing and quality, instead of why testing is part of quality assurance)

5- Distinguish between Error, Defect and Failure (K2)

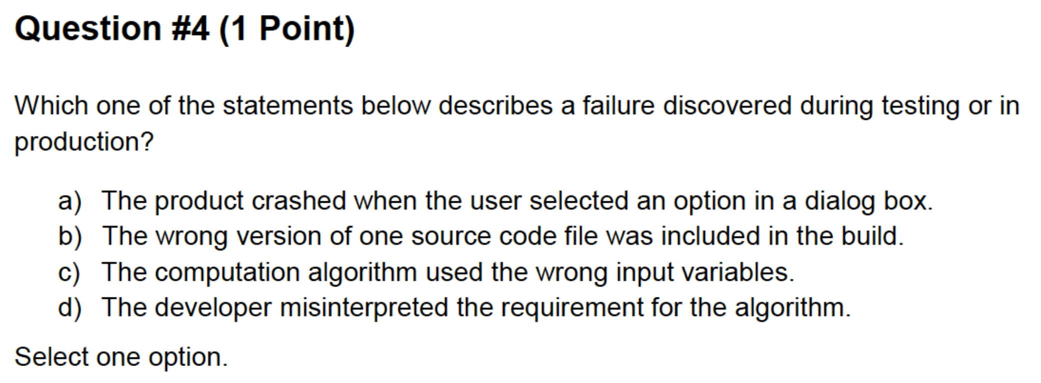
A person can make an error (mistake), which can lead to the introduction of a defect (fault or bug) in the software code or in some other related work product. An error that leads to the introduction of a defect in one work product can trigger an error that leads to the introduction of a defect in a related work product.

For example, a requirements elicitation error can lead to a requirements defect, which then results in a programming error that leads to a defect in the code.

If a defect in the code is executed, this may cause a failure, but not necessarily in all circumstances. For example, some defects require very specific inputs or preconditions to trigger a failure, which may occur rarely or never.

Errors may occur for many reasons, such as:

- Time pressure
- Human fallibility
- Inexperienced or insufficiently skilled project participants
- Miscommunication between project participants, including miscommunication about requirements and design
- Complexity of the code, design, architecture, the underlying problem to be solved, and/or the technologies used
- Misunderstandings about intra-system and inter-system interfaces, especially when such intrasystem and inter-system interactions are large in number
- New, unfamiliar technologies



Question #4 (1 Point)

Which one of the statements below describes a failure discovered during testing or in production?

a) The product crashed when the user selected an option in a dialog box.
b) The wrong version of one source code file was included in the build.
c) The computation algorithm used the wrong input variables.
d) The developer misinterpreted the requirement for the algorithm.

Select one option.

In addition to failures caused due to defects in the code, failures can also be caused by environmental conditions. For example, radiation, electromagnetic fields, and pollution can cause defects in firmware or influence the execution of software by changing hardware conditions.

Not all unexpected test results are failures. False positives may occur due to errors in the way tests were executed, or due to defects in the test data, the test environment, or other testware, or for other reasons.

The inverse situation can also occur, where similar errors or defects lead to false negatives. False negatives are tests that do not detect defects that they should have detected; false positives are reported as defects, but aren't actually defects.

(Fault, mistake, bug removed. Defect is the leading for fault and bug. Error is the leading term for mistake in the glossary)

6- Distinguish between the root cause of a defect and its effects (K2)

The root causes of defects are the earliest actions or conditions that contributed to creating the defects.

Defects can be analyzed to identify their root causes, so as to reduce the occurrence of similar defects in the future. By focusing on the most significant root causes, root cause analysis can lead to process improvements that prevent a significant number of future defects from being introduced.

For example, suppose incorrect interest payments, due to a single line of incorrect code, result in customer complaints. The defective code was written for a user story which was ambiguous, due to the product owner's misunderstanding of how to calculate interest. If a large percentage of defects exist in interest calculations, and these defects have their root cause in similar misunderstandings, the product owners could be trained in the topic of interest calculations to reduce such defects in the future.

In this example, the customer complaints are effects. The incorrect interest payments are failures. The improper calculation in the code is a defect, and it resulted from the original defect, the ambiguity in the user story. The root cause of the original defect was a lack of knowledge on the part of the product owner, which resulted in the product owner making a mistake while writing the user story.

Seven Testing Principles

7- Explain the seven testing principles (K2)

A number of testing principles have been suggested over the past 50 years and offer general guidelines common for all testing.

1. Testing shows the presence of defects, not their absence

Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, testing is not a proof of correctness.

2. Exhaustive testing is impossible

Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Rather than attempting to test exhaustively, risk analysis, test techniques, and priorities should be used to focus test efforts.

3. Early testing saves time and money

To find defects early, both static and dynamic test activities should be started as early as possible in the software development lifecycle. Early testing is sometimes referred to as shift left . Testing early in the software development lifecycle helps reduce or eliminate costly changes (see section 3.1).

4. Defects cluster together

A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures. Predicted defect clusters, and the actual observed defect clusters in test or operation, are an important input into a risk analysis used to focus the test effort (as mentioned in principle 2).

5. Beware of the pesticide paradox

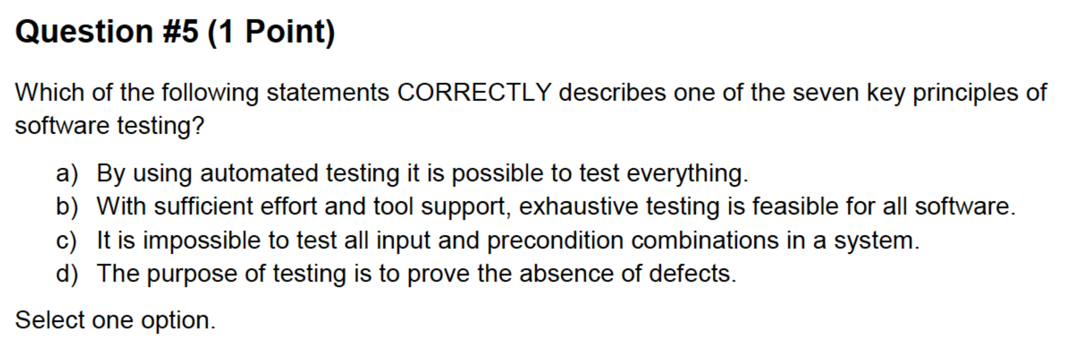
If the same tests are repeated over and over again, eventually these tests no longer find any new defects. To detect new defects, existing tests and test data may need changing, and new tests may need to be written. (Tests are no longer effective at finding defects, just as pesticides are no longer effective at killing insects after a while.) In some cases, such as automated regression testing, the pesticide paradox has a beneficial outcome, which is the relatively low number of regression defects.

6. Testing is context dependent

Testing is done differently in different contexts. For example, safety-critical industrial control software is tested differently from an e-commerce mobile app. As another example, testing in an Agile project is done differently than testing in a sequential lifecycle project (see section 2.1).

7. Absence-of-errors is a fallacy

Some organizations expect that testers can run all possible tests and find all possible defects, but principles 2 and 1, respectively, tell us that this is impossible. Further, it is a fallacy (i.e., a mistaken belief) to expect that just finding and fixing a large number of defects will ensure the success of a system. For example, thoroughly testing all specified requirements and fixing all defects found could still produce a system that is difficult to use, that does not fulfill the users' needs and expectations, or that is inferior compared to other competing systems.



Question #5 (1 Point)

Which of the following statements CORRECTLY describes one of the seven key principles of software testing?

a) By using automated testing it is possible to test everything.
b) With sufficient effort and tool support, exhaustive testing is feasible for all software.
c) It is impossible to test all input and precondition combinations in a system.
d) The purpose of testing is to prove the absence of defects.

Select one option.

Test Process

8- Explain the impact of context on the test process (K2)

There is no one universal software test process, but there are common sets of test activities without which testing will be less likely to achieve its established objectives. These sets of test activities are a test process. The proper, specific software test process in any given situation depends on many factors.

Which test activities are involved in this test process, how these activities are implemented, and when these activities occur may be discussed in an organization's test strategy.

Test Process in Context

Contextual factors that influence the test process for an organization, include, but are not limited to:

- ▶ Software development lifecycle model and project methodologies being used
- ▶ Test levels and test types being considered
- ▶ Product and project risks
- ▶ Business domain
- ▶ Operational constraints, including but not limited to:
    - o Budgets and resources
    - o Timescales
    - o Complexity
    - o Contractual and regulatory requirements
- ▶ Organizational policies and practices
- ▶ Required internal and external standards

The following sections describe general aspects of organizational test processes in terms of the following:

- ▶ Test activities and tasks
- ▶ Test work products
- ▶ Traceability between the test basis and test work products

It is very useful if the test basis (for any level or type of testing that is being considered) has measurable coverage criteria defined. The coverage criteria can act effectively as key performance indicators (KPIs) to drive the activities that demonstrate achievement of software test objectives (see section 1.1.1).

For example, for a mobile application, the test basis may include a list of requirements and a list of supported mobile devices. Each requirement is an element of the test basis. Each supported device is also an element of the test basis. The coverage criteria may require at least one test case for each element of the test basis. Once executed, the results of these tests tell stakeholders whether specified requirements are fulfilled and whether failures were observed on supported devices.

(Testers needs to understand that their testing strategy need to changed based on the context of their project)

9- Describe the test activities and respective tasks within the test process (K2)

A test process consists of the following main groups of activities:

- Test planning
- Test monitoring and control
- Test analysis
- Test design
- Test implementation
- Test execution
- Test completion

Each group of activities is composed of constituent activities, which will be described in the subsections below. Each activity within each group of activities in turn may consist of multiple individual tasks, which would vary from one project or release to another.

Further, although many of these activity groups may appear logically sequential, they are often implemented iteratively. For example, Agile development involves small iterations of software design, build, and test that happen on a continuous basis, supported by on-going planning. So test activities are also happening on an iterative, continuous basis within this development approach. Even in sequential development, the stepped logical sequence of activities will involve overlap, combination, concurrency, or omission, so tailoring these main activities within the context of the system and the project is usually required.

Test planning

Test planning involves activities that define the objectives of testing and the approach for meeting test objectives within constraints imposed by the context (e.g., specifying suitable test techniques and tasks, and formulating a test schedule for meeting a deadline). Test plans may be revisited based on feedback from monitoring and control activities. Test planning is further explained in section 5.2.

Test monitoring and control

Test monitoring involves the on-going comparison of actual progress against the test plan using any test monitoring metrics defined in the test plan. Test control involves taking actions necessary to meet the objectives of the test plan (which may be updated over time). Test monitoring and control are supported by the evaluation of exit criteria, which are referred to as the definition of done in some lifecycles (see ISTQB-AT Foundation Level Agile Tester Extension Syllabus). For example, the evaluation of exit criteria for test execution as part of a given test level may include:

▶ Checking test results and logs against specified coverage criteria

▶ Assessing the level of component or system quality based on test results and logs

▶ Determining if more tests are needed (e.g., if tests originally intended to achieve a certain level of product risk coverage failed to do so, requiring additional tests to be written and executed)

Test progress against the plan is communicated to stakeholders in test progress reports, including deviations from the plan and information to support any decision to stop testing. Test monitoring and control are further explained in section 5.3.

Test analysis

During test analysis, the test basis is analyzed to identify testable features and define associated test conditions. In other words, test analysis determines "what to test" in terms of measurable coverage criteria. Test analysis includes the following major activities:

▶ Analyzing the test basis appropriate to the test level being considered, for example:

- Requirement specifications, such as business requirements, functional requirements, system requirements, user stories, epics, use cases, or similar work products that specify desired functional and non-functional component or system behavior
- Design and implementation information, such as system or software architecture diagrams or documents, design specifications, call flows, modelling diagrams (e.g., UML or entity-relationship diagrams), interface specifications, or similar work products that specify component or system structure
- The implementation of the component or system itself, including code, database metadata and queries, and interfaces
- Risk analysis reports, which may consider functional, non-functional, and structural aspects of the component or system

Question #7 (1 Point)

Which of the below tasks is performed during the test analysis activity of the ISTQB Test Process?

a) Identifying any required infrastructure and tools.
b) Creating test suites from test procedures.
c) Analyzing lessons learned for process improvement.
d) Evaluating the test basis for testability.

Select one option.

▶ Evaluating the test basis and test items to identify defects of various types, such as:

- Ambiguities
- Omissions
- Inconsistencies
- Inaccuracies
- Contradictions
- Superfluous statements

▶ Identifying features and sets of features to be tested

▶ Defining and prioritizing test conditions for each feature based on analysis of the test basis, and considering functional, non-functional, and structural characteristics, other business and technical factors, and levels of risks

▶ Capturing bi-directional traceability between each element of the test basis and the associated test conditions (see sections 1.4.3 and 1.4.4)

The application of black-box, white-box, and experience-based test techniques can be useful in the process of test analysis (see chapter 4) to reduce the likelihood of omitting important test conditions and to define more precise and accurate test conditions.

In some cases, test analysis produces test conditions which are to be used as test objectives in test charters. Test charters are typical work products in some types of experience-based testing (see section 4.4.2). When these test objectives are traceable to the test basis, coverage achieved during such experience-based testing can be measured.

The identification of defects during test analysis is an important potential benefit, especially where no other review process is being used and/or the test process is closely connected with the review process.

Such test analysis activities not only verify whether the requirements are consistent, properly expressed, and complete, but also validate whether the requirements properly capture customer, user, and other stakeholder needs. For example, techniques such as behavior driven development (BDD) and acceptance test driven development (ATDD), which involve generating test conditions and test cases from user stories and acceptance criteria prior to coding, also verify, validate, and detect defects in the user stories and acceptance criteria (see ISTQB Foundation Level Agile Tester Extension syllabus).

Test design

During test design, the test conditions are elaborated into high-level test cases, sets of high-level test cases, and other testware. So, test analysis answers the question “what to test?” while test design answers the question “how to test?”

Test design includes the following major activities:

- Designing and prioritizing test cases and sets of test cases
- Identifying necessary test data to support test conditions and test cases
- Designing the test environment and identifying any required infrastructure and tools
- Capturing bi-directional traceability between the test basis, test conditions, test cases, and test procedures (see section 1.4.4)

The elaboration of test conditions into test cases and sets of test cases during test design often involves using test techniques (see chapter 4).

As with test analysis, test design may also result in the identification of similar types of defects in the test basis. Also as with test analysis, the identification of defects during test design is an important potential benefit.

Test implementation

During test implementation, the testware necessary for test execution is created and/or completed, including sequencing the test cases into test procedures. So, test design answers the question "how to test?" while test implementation answers the question "do we now have everything in place to run the tests?"

Test implementation includes the following major activities:

- Developing and prioritizing test procedures, and, potentially, creating automated test scripts
- Creating test suites from the test procedures and (if any) automated test scripts
- Arranging the test suites within a test execution schedule in a way that results in efficient test execution (see section 5.2.4)
- Building the test environment (including, potentially, test harnesses, service virtualization, simulators, and other infrastructure items) and verifying that everything needed has been set up correctly
- Preparing test data and ensuring it is properly loaded in the test environment
- Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test suites (see section 1.4.4) Test design and test implementation tasks are often combined. In exploratory testing and other types of experience-based testing, test design and implementation may occur, and may be documented, as part of test execution. Exploratory testing may be based on test charters (produced as part of test analysis), and exploratory tests are executed immediately as they are designed and implemented (see section 4.4.2).

Test execution

During test execution, test suites are run in accordance with the test execution schedule.

Test execution includes the following major activities:

- Recording the IDs and versions of the test item(s) or test object, test tool(s), and testware

- Executing tests either manually or by using test execution tools
- Comparing actual results with expected results
- Analyzing anomalies to establish their likely causes (e.g., failures may occur due to defects in the code, but false positives also may occur [see section 1.2.3])
- Reporting defects based on the failures observed (see section 5.6)
- Logging the outcome of test execution (e.g., pass, fail, blocked)
- Repeating test activities either as a result of action taken for an anomaly, or as part of the planned testing (e.g., execution of a corrected test, confirmation testing, and/or regression testing)
- Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test results.

Test completion

Test completion activities collect data from completed test activities to consolidate experience, testware, and any other relevant information. Test completion activities occur at project milestones such as when a software system is released, a test project is completed (or cancelled), an Agile project iteration is finished (e.g., as part of a retrospective meeting), a test level is completed, or a maintenance release has been completed.

Test completion includes the following major activities:

- Checking whether all defect reports are closed, entering change requests or product backlog items for any defects that remain unresolved at the end of test execution
- Creating a test summary report to be communicated to stakeholders
- Finalizing and archiving the test environment, the test data, the test infrastructure, and other testware for later reuse
- Handing over the testware to the maintenance teams, other project teams, and/or other stakeholders who could benefit from its use
- Analyzing lessons learned from the completed test activities to determine changes needed for future iterations, releases, and projects
- Using the information gathered to improve test process maturity

(Updated wording and upgraded from K1 in 2011 to K2 in 2018 Syllabus)

10- Differentiate the work products that support the test process (K2)

Test Work Products

Test work products are created as part of the test process. Just as there is significant variation in the way that organizations implement the test process, there is also significant variation in the types of work products created during that process, in the ways those work products are organized and managed, and in the names used for those work products. This syllabus adheres to the test process outlined above, and the work products described in this syllabus and in the ISTQB Glossary. ISO standard (ISO/IEC/IEEE 29119-3) may also serve as a guideline for test work products.

Many of the test work products described in this section can be captured and managed using test management tools and defect management tools (see chapter 6).

Test planning work products

Test planning work products typically include one or more test plans. The test plan includes information about the test basis, to which the other test work products will be related via traceability information (see below and section 1.4.4), as well as exit criteria (or definition of done) which will be used during test monitoring and control. Test plans are described in section 5.2.

Test monitoring and control work products

Test monitoring and control work products typically include various types of test reports, including test progress reports (produced on an ongoing and/or a regular basis) and test summary reports (produced at various completion milestones). All test reports should provide audience-relevant details about the test progress as of the date of the report, including summarizing the test execution results once those become available.

Test monitoring and control work products should also address project management concerns, such as task completion, resource allocation and usage, and effort.

Test monitoring and control, and the work products created during these activities, are further explained in section 5.3 of this syllabus.

Test analysis work products

Test analysis work products include defined and prioritized test conditions, each of which is ideally bidirectionally traceable to the specific element(s) of the test basis it covers. For exploratory testing, test analysis may involve the creation of test charters. Test analysis may also result in the discovery and reporting of defects in the test basis.

Test design work products

Test design results in test cases and sets of test cases to exercise the test conditions defined in test analysis. It is often a good practice to design high-level test cases, without concrete values for input data and expected results. Such high-level test cases are reusable across multiple test cycles with different concrete data, while still adequately documenting the scope of the test case. Ideally, each test case is bidirectionally traceable to the test condition(s) it covers.

Test design also results in the design and/or identification of the necessary test data, the design of the test environment, and the identification of infrastructure and tools, though the extent to which these results are documented varies significantly. Test conditions defined in test analysis may be further refined in test design.

Test implementation work products

Test implementation work products include:

- ▶ Test procedures and the sequencing of those test procedures
- ▶ Test suites
- ▶ A test execution schedule

Ideally, once test implementation is complete, achievement of coverage criteria established in the test plan can be demonstrated via bi-directional traceability between test procedures and specific elements of the test basis, through the test cases and test conditions.

In some cases, test implementation involves creating work products using or used by tools, such as service virtualization and automated test scripts.

Test implementation also may result in the creation and verification of test data and the test environment.

The completeness of the documentation of the data and/or environment verification results may vary significantly.

The test data serve to assign concrete values to the inputs and expected results of test cases. Such concrete values, together with explicit directions about the use of the concrete values, turn high-level test cases into executable low-level test cases. The same high-level test case may use different test data when executed on different releases of the test object. The concrete expected results which are associated with concrete test data are identified by using a test oracle.

In exploratory testing, some test design and implementation work products may be created during test execution, though the extent to which exploratory tests (and their traceability to specific elements of the test basis) are documented may vary significantly.

Test conditions defined in test analysis may be further refined in test implementation.

Test execution work products

Test execution work products include:

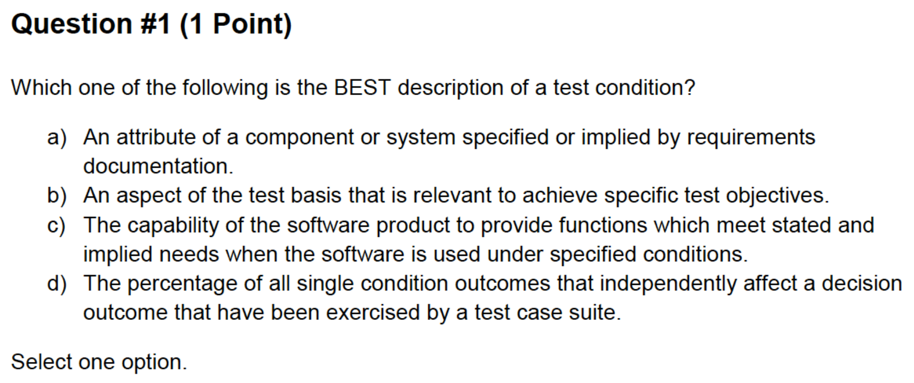
- ▶ Documentation of the status of individual test cases or test procedures (e.g., ready to run, pass, fail, blocked, deliberately skipped, etc.)
- ▶ Defect reports (see section 5.6)
- ▶ Documentation about which test item(s), test object(s), test tools, and testware were involved in the testing

Ideally, once test execution is complete, the status of each element of the test basis can be determined and reported via bi-directional traceability with the associated the test procedure(s). For example, we can say which requirements have passed all planned tests, which requirements have failed tests and/or have defects associated with them, and which requirements have planned tests still waiting to be run. This enables verification that the coverage criteria have been met, and enables the reporting of test results in terms that are understandable to stakeholders.

Test completion work products

Test completion work products include test summary reports, action items for improvement of subsequent projects or iterations (e.g., following a project Agile retrospective), change requests or product backlog items, and finalized testware.

(Moved from section 4.1 updated wording to accomodate better exam question)

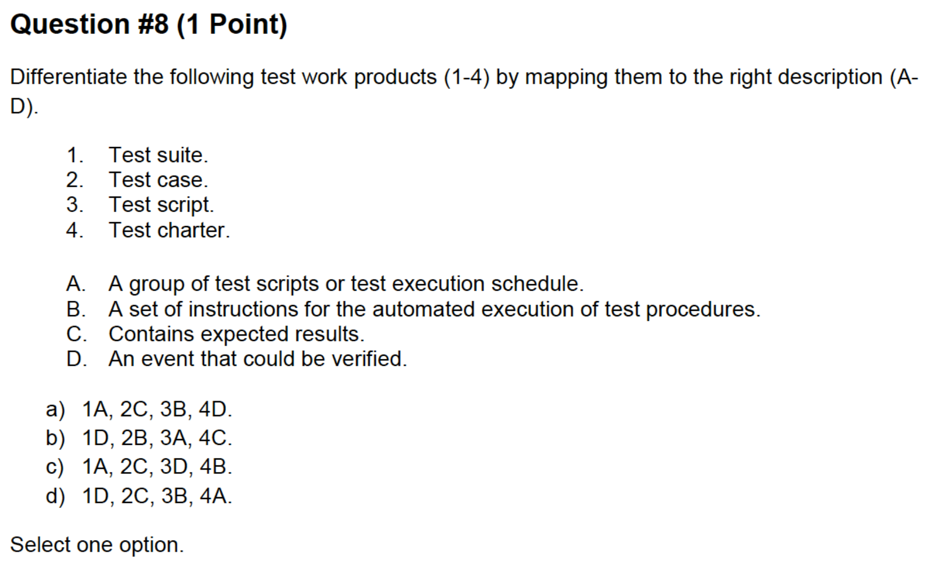


Question #1 (1 Point)

Which one of the following is the BEST description of a test condition?

a) An attribute of a component or system specified or implied by requirements documentation.
b) An aspect of the test basis that is relevant to achieve specific test objectives.
c) The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
d) The percentage of all single condition outcomes that independently affect a decision outcome that have been exercised by a test case suite.

Select one option.



Question #8 (1 Point)

Differentiate the following test work products (1-4) by mapping them to the right description (A-D).

1. Test suite.
2. Test case.
3. Test script.
4. Test charter.

A. A group of test scripts or test execution schedule.
B. A set of instructions for the automated execution of test procedures.
C. Contains expected results.
D. An event that could be verified.

a) 1A, 2C, 3B, 4D.
b) 1D, 2B, 3A, 4C.
c) 1A, 2C, 3D, 4B.
d) 1D, 2C, 3B, 4A.

Select one option.

11- Explain the value of maintaining traceability between the test basis and the test work products (K2)

As mentioned in section 1.4.3, test work products and the names of those work products vary significantly. Regardless of these variations, in order to implement effective test monitoring and control, it is important to establish and maintain traceability throughout the test process between each element of the test basis and the various test work products associated with that element, as described above. In addition to the evaluation of test coverage, good traceability supports:

- Analyzing the impact of changes
- Making testing auditable
- Meeting IT governance criteria
- Improving the understandability of test progress reports and test summary reports to include the status of elements of the test basis (e.g., requirements that passed their tests, requirements that failed their tests, and requirements that have pending tests)
- Relating the technical aspects of testing to stakeholders in terms that they can understand
- Providing information to assess product quality, process capability, and project progress against business goals

Some test management tools provide test work product models that match part or all of the test work products outlined in this section. Some organizations build their own management systems to organize the work products and provide the information traceability they require.

(Moved from section 4.1 updated wording to accomodate better exam question)

The Psychology of Testing

12- Identify the psychological factors that influence the success of testing (K1)

Identifying defects during a static test such as a requirements review or user story refinement session, or identifying failures during dynamic test execution, may be perceived as criticism of the product and of its author. An element of human psychology called confirmation bias can make it difficult to accept information that disagrees with currently held beliefs. For example, since developers expect their code to be correct, they have a confirmation bias that makes it difficult to accept that the code is incorrect. In addition to confirmation bias, other cognitive biases may make it difficult for people to understand or accept information produced by testing. Further, it is a common human trait to blame the bearer of bad news, and information produced by testing often contains bad news.

As a result of these psychological factors, some people may perceive testing as a destructive activity, even though it contributes greatly to project progress and product quality (see sections 1.1 and 1.2). To try to reduce these perceptions, information about defects and failures should be communicated in a constructive way. This way, tensions between the testers and the analysts, product owners, designers, and developers can be reduced. This applies during both static and dynamic testing.

Testers and test managers need to have good interpersonal skills to be able to communicate effectively about defects, failures, test results, test progress, and risks, and to build positive relationships with colleagues. Ways to communicate well include the following examples:

- Start with collaboration rather than battles. Remind everyone of the common goal of better quality systems.
- Emphasize the benefits of testing. For example, for the authors, defect information can help them improve their work products and their skills. For the organization, defects found and fixed during testing will save time and money and reduce overall risk to product quality.

- ▶ Communicate test results and other findings in a neutral, fact-focused way without criticizing the person who created the defective item. Write objective and factual defect reports and review findings.
- ▶ Try to understand how the other person feels and the reasons they may react negatively to the information.
- ▶ Confirm that the other person has understood what has been said and vice versa.

Typical test objectives were discussed earlier (see section 1.1). Clearly defining the right set of test objectives has important psychological implications. Most people tend to align their plans and behaviors with the objectives set by the team, management, and other stakeholders. It is also important that testers adhere to these objectives with minimal personal bias.

(Identify is more evidence based wording)

13- Explain the difference between the mindset required for test activities and the mindset required for development activities (K2)

Developers and testers often think differently. The primary objective of development is to design and build a product. As discussed earlier, the objectives of testing include verifying and validating the product, finding defects prior to release, and so forth. These are different sets of objectives which require different mindsets. Bringing these mindsets together helps to achieve a higher level of product quality.

A mindset reflects an individual's assumptions and preferred methods for decision making and problem solving. A tester's mindset should include curiosity, professional pessimism, a critical eye, attention to detail, and a motivation for good and positive communications and relationships. A tester's mindset tends to grow and mature as the tester gains experience.

A developer's mindset may include some of the elements of a tester's mindset, but successful developers are often more interested in designing and building solutions than in contemplating what might be wrong with those solutions. In addition, confirmation bias makes it difficult to find mistakes in their own work.

With the right mindset, developers are able to test their own code. Different software development lifecycle models often have different ways of organizing the testers and test activities. Having some of the test activities done by independent testers increases defect detection effectiveness, which is particularly important for large, complex, or safety-critical systems. Independent testers bring a perspective which is different than that of the work product authors (i.e., business analysts, product owners, designers, and programmers), since they have different cognitive biases from the authors.

(Rephrased for better understandability)

Chapter 2 Testing Throughout the Software Development Lifecycle

Software Development Lifecycle Models

14- Explain relationships between software development activities and test activities in the software development lifecycle (K2)

A software development lifecycle model describes the types of activity performed at each stage in a software development project, and how the activities relate to one another logically and chronologically.

There are a number of different software development lifecycle models, each of which requires different approaches to testing.

Software Development and Software Testing

It is an important part of a tester's role to be familiar with the common software development lifecycle models so that appropriate test activities can take place.

In any software development lifecycle model, there are several characteristics of good testing:

- For every development activity, there is a corresponding test activity
- Each test level has test objectives specific to that level
- Test analysis and design for a given test level begin during the corresponding development activity
- Testers participate in discussions to define and refine requirements and design, and are involved in reviewing work products (e.g., requirements, design, user stories, etc.) as soon as drafts are available

No matter which software development lifecycle model is chosen, test activities should start in the early stages of the lifecycle, adhering to the testing principle of early testing.

This syllabus categorizes common software development lifecycle models as follows:

- Sequential development models
- Iterative and incremental development models

A sequential development model describes the software development process as a linear, sequential flow of activities. This means that any phase in the development process should begin when the previous phase is complete. In theory, there is no overlap of phases, but in practice, it is beneficial to have early feedback from the following phase.

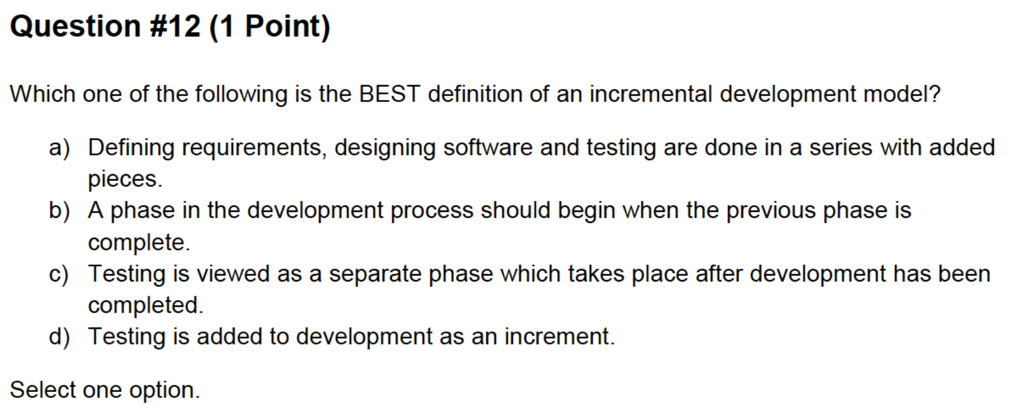
In the Waterfall model, the development activities (e.g., requirements analysis, design, coding, testing) are completed one after another. In this model, test activities only occur after all other development activities have been completed.

Unlike the Waterfall model, the V-model integrates the test process throughout the development process, implementing the principle of early testing. Further, the V-model includes test levels associated with each corresponding development phase, which further supports early testing (see section 2.2 for a discussion of test levels). In this model, the execution of tests associated with each test level proceeds sequentially, but in some cases overlapping occurs.

Sequential development models deliver software that contains the complete set of features, but typically require months or years for delivery to stakeholders and users.

Incremental development involves establishing requirements, designing, building, and testing a system in pieces, which means that the software's features grow incrementally. The size of these feature increments vary, with some methods having larger pieces and some smaller pieces. The feature increments can be as small as a single change to a user interface screen or new query option.

Iterative development occurs when groups of features are specified, designed, built, and tested together in a series of cycles, often of a fixed duration. Iterations may involve changes to features developed in earlier iterations, along with changes in project scope. Each iteration delivers working software which is a growing subset of the overall set of features until the final software is delivered or development is stopped.



Question #12 (1 Point)

Which one of the following is the BEST definition of an incremental development model?

a) Defining requirements, designing software and testing are done in a series with added pieces.
b) A phase in the development process should begin when the previous phase is complete.
c) Testing is viewed as a separate phase which takes place after development has been completed.
d) Testing is added to development as an increment.

Select one option.

Examples include:

▶ Rational Unified Process: Each iteration tends to be relatively long (e.g., two to three months), and the feature increments are correspondingly large, such as two or three groups of related features

▶ Scrum: Each iteration tends to be relatively short (e.g., hours, days, or a few weeks), and the feature increments are correspondingly small, such as a few enhancements and/or two or three new features

▶ Kanban: Implemented with or without fixed-length iterations, which can deliver either a single enhancement or feature upon completion, or can group features together to release at once

▶ Spiral (or prototyping): Involves creating experimental increments, some of which may be heavily re-worked or even abandoned in subsequent development work

Components or systems developed using these methods often involve overlapping and iterating test levels throughout development. Ideally, each feature is tested at several test levels as it moves towards delivery. In some cases, teams use continuous delivery or continuous deployment, both of which involve significant automation of multiple test levels as part of their delivery pipelines. Many development efforts using these methods also include the concept of self-organizing teams, which can change the way testing work is organized as well as the relationship between testers and developers.

These methods form a growing system, which may be released to end-users on a feature-by-feature basis, on an iteration-by-iteration basis, or in a more traditional major-release fashion. Regardless of whether the software increments are released to end-users, regression testing is increasingly important as the system grows.

In contrast to sequential models, iterative and incremental models may deliver usable software in weeks or even days, but may only deliver the complete set of requirements product over a period of months or even years.

(Text is simplified and reduced)

15- Identify reasons why software development lifecycle models must be adapted to the context of project and product characteristics (K1)

Software development lifecycle models must be selected and adapted to the context of project and product characteristics. An appropriate software development lifecycle model should be selected and adapted based on the project goal, the type of product being developed, business priorities (e.g., time-tomarket), and identified product and project risks. For example, the development and testing of a minor internal administrative system should differ from the development and testing of a safety-critical system such as an automobile's brake control system. As another example, in some cases organizational and cultural issues may inhibit communication between team members, which can impede iterative development.

Depending on the context of the project, it may be necessary to combine or reorganize test levels and/or test activities. For example, for the integration of a commercial off-the-shelf (COTS) software product into a larger system, the purchaser may perform interoperability testing at the system integration test level (e.g., integration to the infrastructure and other systems) and at the acceptance test level (functional and non-functional, along with user acceptance testing and operational acceptance testing). See section 2.2 for a discussion of test levels and section 2.3 for a discussion of test types.

In addition, software development lifecycle models themselves may be combined. For example, a Vmodel may be used for the development and testing of the backend systems and their integrations, while an Agile development model may be used to develop and test the front-end user interface (UI) and functionality. Prototyping may be used early in a project, with an incremental development model adopted once the experimental phase is complete. Internet of Things (IoT) systems, which consist of many different objects, such as devices, products, and services, typically apply separate software development lifecycle models for each object. This presents a particular challenge for the development of Internet of Things system versions. Additionally the software development lifecycle of such objects places stronger emphasis on the later phases of the software development lifecycle after they have been introduced to operational use (e.g., operate, update, and decommission phases).

(Identify is more evidence based wording)

Test Levels

16- Compare the different test levels from the perspective of objective, test basis, test objects, typical defects and failures and approaches and responsibilities. (K2)

Test levels are groups of test activities that are organized and managed together. Each test level is an instance of the test process, consisting of the activities described in section 1.4, performed in relation to software at a given level of development, from individual units or components to complete systems or, where applicable, systems of systems. Test levels are related to other activities within the software development lifecycle. The test levels used in this syllabus are:

- ▶ Component testing
- ▶ Integration testing
- ▶ System testing
- ▶ Acceptance testing

Test levels are characterized by the following attributes:

- ▶ Specific objectives
- ▶ Test basis, referenced to derive test cases
- ▶ Test object (i.e., what is being tested)
- ▶ Typical defects and failures
- ▶ Specific approaches and responsibilities

For every test level, a suitable test environment is required. In acceptance testing, for example, a production-like test environment is ideal, while in component testing the developers typically use their own development environment.

Question #10 (1 Point)

Which of the following statements comparing component testing and system testing is TRUE?

a) Component testing verifies the functionality of software modules, program objects, and classes that are separately testable, whereas system testing verifies interfaces between components and interactions between different parts of the system.
b) Test cases for component testing are usually derived from component specifications, design specifications, or data models, whereas test cases for system testing are usually derived from requirement specifications, functional specifications, or use cases.
c) Component testing only focuses on functional characteristics, whereas system testing focuses on functional and non-functional characteristics.
d) Component testing is the responsibility of the testers, whereas system testing typically is the responsibility of the users of the system.

Select one option.

	Objectives	Test basis	Test objects	Typical defects and failures	Specific approaches and responsibilities
Component Testing	Component testing (also known as unit or module testing) focuses on components that are separately testable. Objectives of component testing include: ▶ Reducing risk ▶ Verifying whether the functional and non-functional behaviors of the component are as designed and specified ▶ Building confidence in the component's quality ▶ Finding defects in the component ▶ Preventing defects from escaping to higher test levels In some cases, especially in incremental and iterative development models (e.g., Agile) where code changes are ongoing, automated component regression tests play a key role in building confidence that changes have not broken existing components. Component testing is often done in isolation from the rest of the system, depending on the software development lifecycle model and the system, which may require mock objects, service virtualization, harnesses, stubs, and drivers. Component testing may cover functionality (e.g., correctness of calculations), non-functional characteristics (e.g., searching for memory leaks), and structural properties (e.g., decision testing).	Examples of work products that can be used as a test basis for component testing include: ▶ Detailed design ▶ Code ▶ Data model ▶ Component specifications	Typical test objects for component testing include: ▶ Components, units or modules ▶ Code and data structures ▶ Classes ▶ Database modules	Examples of typical defects and failures for component testing include: ▶ Incorrect functionality (e.g., not as described in design specifications) ▶ Data flow problems ▶ Incorrect code and logic Defects are typically fixed as soon as they are found, often with no formal defect management. However, when developers do report defects, this provides important information for root cause analysis and process improvement.	Component testing is usually performed by the developer who wrote the code, but it at least requires access to the code being tested. Developers may alternate component development with finding and fixing defects. Developers will often write and execute tests after having written the code for a component. However, in Agile development especially, writing automated component test cases may precede writing application code. For example, consider test driven development (TDD). Test driven development is highly iterative and is based on cycles of developing automated test cases, then building and integrating small pieces of code, then executing the component tests, correcting any issues, and re-factoring the code. This process continues until the component has been completely built and all component tests are passing. Test driven development is an example of a test-first approach. While test driven development originated in eXtreme Programming (XP), it has spread to other forms of Agile and also to sequential lifecycles (see ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

	Objectives	Test basis	Test objects	Typical defects and failures	Specific approaches and responsibilities
Integration testing	Integration testing focuses on interactions between components or systems. Objectives of integration testing include: ▶ Reducing risk ▶ Verifying whether the functional and non-functional behaviors of the interfaces are as designed and specified ▶ Building confidence in the quality of the interfaces ▶ Finding defects (which may be in the interfaces themselves or within the components or systems) ▶ Preventing defects from escaping to higher test levels As with component testing, in some cases automated integration regression tests provide confidence that changes have not broken existing interfaces, components, or systems. There are two different levels of integration testing described in this syllabus, which may be carried out on test objects of varying size as follows: ▶ Component integration testing focuses on the interactions and interfaces between integrated components. Component integration testing is performed after component testing, and is generally automated. In iterative and incremental development, component integration tests are usually part of the continuous integration process. ▶ System integration testing focuses on the interactions and interfaces between systems, packages, and microservices. System integration testing can also cover interactions with, and interfaces provided by, external organizations (e.g., web services). In this case, the developing organization does not control the external interfaces, which can create various challenges for testing (e.g., ensuring that test-blocking defects in the external organization's code are resolved, arranging for test environments, etc.). System integration testing may be done after system testing or in parallel with ongoing system test activities (in both sequential development and iterative and incremental development).	Examples of work products that can be used as a test basis for integration testing include: ▶ Software and system design ▶ Sequence diagrams ▶ Interface and communication protocol specifications ▶ Use cases ▶ Architecture at component or system level ▶ Workflows ▶ External interface definitions	Typical test objects for integration testing include: ▶ Subsystems ▶ Databases ▶ Infrastructure ▶ Interfaces ▶ APIs ▶ Microservices	Examples of typical defects and failures for component integration testing include: ▶ Incorrect data, missing data, or incorrect data encoding ▶ Incorrect sequencing or timing of interface calls ▶ Interface mismatch ▶ Failures in communication between components ▶ Unhandled or improperly handled communication failures between components ▶ Incorrect assumptions about the meaning, units, or boundaries of the data being passed between components Examples of typical defects and failures for system integration testing include: ▶ Inconsistent message structures between systems ▶ Incorrect data, missing data, or incorrect data encoding ▶ Incorrect assumptions about the meaning, units, or boundaries of the data being passed between systems ▶ Failure to comply with mandatory security regulations ▶ Interface mismatch ▶ Failures in communication between systems ▶ Unhandled or improperly handled communication failures between systems	Component integration tests and system integration tests should concentrate on the integration itself. For example, if integrating module A with module B, tests should focus on the communication between the modules, not the functionality of the individual modules, as that should have been covered during component testing. If integrating system X with system Y, tests should focus on the communication between the systems, not the functionality of the individual systems, as that should have been covered during system testing. Functional, non-functional, and structural test types are applicable. Component integration testing is often the responsibility of developers. System integration testing is generally the responsibility of testers. Ideally, testers performing system integration testing should understand the system architecture, and should have influenced integration planning. If integration tests and the integration strategy are planned before components or systems are built, those components or systems can be built in the order required for most efficient testing. Systematic integration strategies may be based on the system architecture (e.g., top-down and bottom-up), functional tasks, transaction processing sequences, or some other aspect of the system or components. In order to simplify defect isolation and detect defects early, integration should normally be incremental (i.e., a small number of additional components or systems at a time) rather than "big bang" (i.e., integrating all components or systems in one single step). A risk analysis of the most complex interfaces can help to focus the integration testing. The greater the scope of integration, the more difficult it becomes to isolate defects to a specific component or system, which may lead to increased risk and additional time for troubleshooting. This is one reason that continuous integration, where software is integrated on a component-by-component basis (i.e., functional integration), has become common practice. Such continuous integration often includes automated regression testing, ideally at multiple test levels.

	Objectives	Test basis	Test objects	Typical defects and failures	Specific approaches and responsibilities
System testing	System testing focuses on the behavior and capabilities of a whole system or product, often considering the end-to-end tasks the system can perform and the non-functional behaviors it exhibits while performing those tasks. Objectives of system testing include: ▶ Reducing risk ▶ Verifying whether the functional and non-functional behaviors of the system are as designed and specified ▶ Validating that the system is complete and will work as expected ▶ Building confidence in the quality of the system as a whole ▶ Finding defects ▶ Preventing defects from escaping to higher test levels or production For certain systems, verifying data quality may be an objective. As with component testing and integration testing, in some cases automated system regression tests provide confidence that changes have not broken existing features or end-to-end capabilities. System testing often produces information that is used by stakeholders to make release decisions. System testing may also satisfy legal or regulatory requirements or standards. The test environment should ideally correspond to the final target or production environment	Examples of work products that can be used as a test basis for system testing include: ▶ System and software requirement specifications (functional and non-functional) ▶ Risk analysis reports ▶ Use cases ▶ Epics and user stories ▶ Models of system behavior ▶ State diagrams ▶ System and user manuals	Typical test objects for system testing include: ▶ Applications ▶ Hardware/ software systems ▶ Operating systems ▶ System under test (SUT) ▶ System configuration and configuration data	Examples of typical defects and failures for system testing include: ▶ Incorrect calculations ▶ Incorrect or unexpected system functional or non-functional behavior ▶ Incorrect control and/or data flows within the system ▶ Failure to properly and completely carry out end-to-end functional tasks ▶ Failure of the system to work properly in the production environment(s) ▶ Failure of the system to work as described in system and user manuals	System testing should focus on the overall, end-to-end behavior of the system as a whole, both functional and non-functional. System testing should use the most appropriate techniques (see chapter 4) for the aspect(s) of the system to be tested. For example, a decision table may be created to verify whether functional behavior is as described in business rules. Independent testers typically carry out system testing. Defects in specifications (e.g., missing user stories, incorrectly stated business requirements, etc.) can lead to a lack of understanding of, or disagreements about, expected system behavior. Such situations can cause false positives and false negatives, which waste time and reduce defect detection effectiveness, respectively. Early involvement of testers in user story refinement or static testing activities, such as reviews, helps to reduce the incidence of such situations.

	Objectives	Test basis	Test objects	Typical defects and failures	Specific approaches and responsibilities
Acceptance testing	Acceptance testing, like system testing, typically focuses on the behavior and capabilities of a whole system or product. Objectives of acceptance testing include: ▶ Establishing confidence in the quality of the system as a whole ▶ Validating that the system is complete and will work as expected ▶ Verifying that functional and non-functional behaviors of the system are as specified Acceptance testing may produce information to assess the system's readiness for deployment and use by the customer (end-user). Defects may be found during acceptance testing, but finding defects is often not an objective, and finding a significant number of defects during acceptance testing may in some cases be considered a major project risk. Acceptance testing may also satisfy legal or regulatory requirements or standards. Common forms of acceptance testing include the following: ▶ User acceptance testing ▶ Operational acceptance testing ▶ Contractual and regulatory acceptance testing ▶ Alpha and beta testing. Each is described in the following four subsections.	Examples of work products that can be used as a test basis for any form of acceptance testing include: ▶ Business processes ▶ User or business requirements ▶ Regulations, legal contracts and standards ▶ Use cases ▶ System requirements ▶ System or user documentation ▶ Installation procedures ▶ Risk analysis reports In addition, as a test basis for deriving test cases for operational acceptance testing, one or more of the following work products can be used: ▶ Backup and restore procedures ▶ Disaster recovery procedures ▶ Non-functional requirements ▶ Operations documentation ▶ Deployment and installation instructions ▶ Performance targets ▶ Database packages ▶ Security standards or regulations	Typical test objects for any form of acceptance testing include: ▶ System under test ▶ System configuration and configuration data ▶ Business processes for a fully integrated system ▶ Recovery systems and hot sites (for business continuity and disaster recovery testing) ▶ Operational and maintenance processes ▶ Forms ▶ Reports ▶ Existing and converted production data	Examples of typical defects for any form of acceptance testing include: ▶ System workflows do not meet business or user requirements ▶ Business rules are not implemented correctly ▶ System does not satisfy contractual or regulatory requirements ▶ Non-functional failures such as security vulnerabilities, inadequate performance efficiency under high loads, or improper operation on a supported platform	Specific approaches and responsibilities Acceptance testing is often the responsibility of the customers, business users, product owners, or operators of a system, and other stakeholders may be involved as well. Acceptance testing is often thought of as the last test level in a sequential development lifecycle, but it may also occur at other times, for example: ▶ Acceptance testing of a COTS software product may occur when it is installed or integrated ▶ Acceptance testing of a new functional enhancement may occur before system testing In iterative development, project teams can employ various forms of acceptance testing during and at the end of each iteration, such as those focused on verifying a new feature against its acceptance criteria and those focused on validating that a new feature satisfies the users' needs. In addition, alpha tests and beta tests may occur, either at the end of each iteration, after the completion of each iteration, or after a series of iterations. User acceptance tests, operational acceptance tests, regulatory acceptance tests, and contractual acceptance tests also may occur, either at the close of each iteration, after the completion of each iteration, or after a series of iterations.

Acceptance Test Types

User acceptance testing (UAT)

The acceptance testing of the system by users is typically focused on validating the fitness for use of the system by intended users in a real or simulated operational environment. The main objective is building confidence that the users can use the system to meet their needs, fulfill requirements, and perform business processes with minimum difficulty, cost, and risk.

Operational acceptance testing (OAT)

The acceptance testing of the system by operations or systems administration staff is usually performed in a (simulated) production environment. The tests focus on operational aspects, and may include:

- Testing of backup and restore
- Installing, uninstalling and upgrading
- Disaster recovery
- User management
- Maintenance tasks
- Data load and migration tasks
- Checks for security vulnerabilities
- Performance testing

The main objective of operational acceptance testing is building confidence that the operators or system administrators can keep the system working properly for the users in the operational environment, even under exceptional or difficult conditions.

Contractual and regulatory acceptance testing

Contractual acceptance testing is performed against a contract's acceptance criteria for producing custom-developed software. Acceptance criteria should be defined when the parties agree to the contract. Contractual acceptance testing is often performed by users or by independent testers.

Regulatory acceptance testing is performed against any regulations that must be adhered to, such as government, legal, or safety regulations. Regulatory acceptance testing is often performed by users or by independent testers, sometimes with the results being witnessed or audited by regulatory agencies.

The main objective of contractual and regulatory acceptance testing is building confidence that contractual or regulatory compliance has been achieved.

Alpha and beta testing

Alpha and beta testing are typically used by developers of commercial off-the-shelf (COTS) software who want to get feedback from potential or existing users, customers, and/or operators before the software product is put on the market.

Alpha testing is performed at the developing organization's site, not by the development team, but by potential or existing customers, and/or operators or an independent test team.

Beta testing is performed by potential or existing customers, and/or operators at their own locations. Beta testing may come after alpha testing, or may occur without any preceding alpha testing having occurred.

One objective of alpha and beta testing is building confidence among potential or existing customers, and/or operators that they can use the system under normal, everyday conditions, and in the operational environment(s) to achieve their objectives with minimum difficulty, cost, and risk. Another objective may be the detection of defects related to the conditions and environment(s) in which the system will be used, especially when those conditions and environment(s) are difficult to replicate by the development team.

(Responsibilities is what is to know and understand, instead of "people who test". Evidence of mastering this LO is better to get in certifications)

Test Types

17- Compare functional, non-functional and white-box testing (K2)

Functional Testing

Functional testing of a system involves tests that evaluate functions that the system should perform. Functional requirements may be described in work products such as business requirements specifications, epics, user stories, use cases, or functional specifications, or they may be undocumented. The functions are "what" the system should do.

Functional tests should be performed at all test levels (e.g., tests for components may be based on a component specification), though the focus is different at each level (see section 2.2).

Functional testing considers the behavior of the software, so black-box techniques may be used to derive test conditions and test cases for the functionality of the component or system (see section 4.2).

The thoroughness of functional testing can be measured through functional coverage. Functional coverage is the extent to which some type of functional element has been exercised by tests, and is expressed as a percentage of the type(s) of element being covered. For example, using traceability between tests and functional requirements, the percentage of these requirements which are addressed by testing can be calculated, potentially identifying coverage gaps.

Functional test design and execution may involve special skills or knowledge, such as knowledge of the particular business problem the software solves (e.g., geological modelling software for the oil and gas industries) or the particular role the software serves (e.g., computer gaming software that provides interactive entertainment).

Non-functional Testing

Non-functional testing of a system evaluates characteristics of systems and software such as usability, performance efficiency or security. Refer to ISO standard (ISO/IEC 25010) for a classification of software product quality characteristics. Non-functional testing is the testing of "how well" the system behaves.

Contrary to common misperceptions, non-functional testing can and often should be performed at all test levels, and done as early as possible. The late discovery of non-functional defects can be extremely dangerous to the success of a project. Black-box techniques (see section 4.2) may be used to derive test conditions and test cases for nonfunctional testing. For example, boundary value analysis can be used to define the stress conditions for performance tests.

The thoroughness of non-functional testing can be measured through non-functional coverage. Nonfunctional coverage is the extent to which some type of non-functional element has been exercised by tests, and is expressed as a percentage of the type(s) of element being covered. For example, using traceability between tests and supported devices for a mobile application, the percentage of devices which are addressed by compatibility testing can be calculated, potentially identifying coverage gaps.

Non-functional test design and execution may involve special skills or knowledge, such as knowledge of the inherent weaknesses of a design or technology (e.g., security vulnerabilities associated with particular programming languages) or the particular user base (e.g., the personas of users of healthcare facility management systems).

White-box Testing

White-box testing derives tests based on the system's internal structure or implementation. Internal structure may include code, architecture, work flows, and/or data flows within the system (see section 4.3).

The thoroughness of white-box testing can be measured through structural coverage. Structural coverage is the extent to which some type of structural element has been exercised by tests, and is expressed as a percentage of the type of element being covered.

At the component testing level, code coverage is based on the percentage of component code that has been tested, and may be measured in terms of different aspects of code (coverage items) such as the percentage of executable statements tested in the component, or the percentage of decision outcomes tested. These types of coverage are collectively called code coverage. At the component integration testing level, white-box testing may be based on the architecture of the system, such as interfaces between components, and structural coverage may be measured in terms of the percentage of interfaces exercised by tests.

White-box test design and execution may involve special skills or knowledge, such as the way the code is built (e.g., to use code coverage tools), how data is stored (e.g., to evaluate possible database queries), and how to use coverage tools and to correctly interpret their results.

(Better understandability. Structural testing is changed to white-box testing to make it easier on Foundation level and at the same time ensure alignment with ISTQB advanced level use of white-box.)

18- Recognize that functional, non-functional and white-box tests occur at any test level (K1)

TEST LEVELS/TEST TYPES (based on a bank application)				
	FUNCTIONAL	NON-FUNCTIONAL	WHITE-BOX	CHANGE-RELATED
Component Testing	Tests are based on how a component should calculate compound interest	**Performance tests:** Designed to evaluate the number of CPU cycles required to perform a complex total interest calculation.	Tests designed to achieve complete statement and decision coverage for all components that perform financial calculations.	**Automated regression tests:** Tests are built for each component and included within the continuous integration framework.
Component-Integration Testing	Tests are based on how account information captured at the user interface is passed to the business logic.	**Security tests:** Designed for buffer overflow vulnerabilities due to data passed from the user interface to the business logic.	Tests designed to exercise how each screen in the browser interface passes data to the next screen and to the business logic.	Tests designed to confirm fixes to interface-related defects as the fixes are checked into the code repository.
System Testing	Tests are based on how account holders can apply for a line of credit on their checking accounts.	**Portability tests:** Designed to check whether the presentation layer works on all supported browsers and mobile devices.	Tests designed to cover sequences of web pages that can occur during a credit line application.	All tests for a given workflow are re-executed if any screen on that workflow changes.
System Integration Testing	Tests are based on how the system uses an external micro service to check an account holder's credit score.	**Reliability tests:** Designed to evaluate system robustness if the credit score micro service fails to respond.	Tests designed to exercise all possible inquiry types sent to the credit score microservice.	Tests of the application interacting with the credit scoring micro service are re-executed daily as part of continuous deployment of that microservice.
Acceptance Testing	Tests are based on how the banker handles approving or declining a credit application	**Usability tests:** Designed to evaluate the accessibility of the banker's credit processing interface for people with disabilities.	Tests designed to cover all supported financial data file structures and value ranges for bank-to-bank transfers.	All previously-failed tests are re-executed after a defect found in acceptance testing is fixed.

Question #9 (1 Point)

How can white-box testing be applied during acceptance testing?

a) To check if large volumes of data can be transferred between integrated systems.
b) To check if all code statements and code decision paths have been executed.
c) To check if all work process flows have been covered.
d) To cover all web page navigations.

Select one option.

19- Compare the Purposes of Confirmation testing and Regression testing (K2)

When changes are made to a system, either to correct a defect or because of new or changing functionality, testing should be done to confirm that the changes have corrected the defect or implemented the functionality correctly, and have not caused any unforeseen adverse consequences.

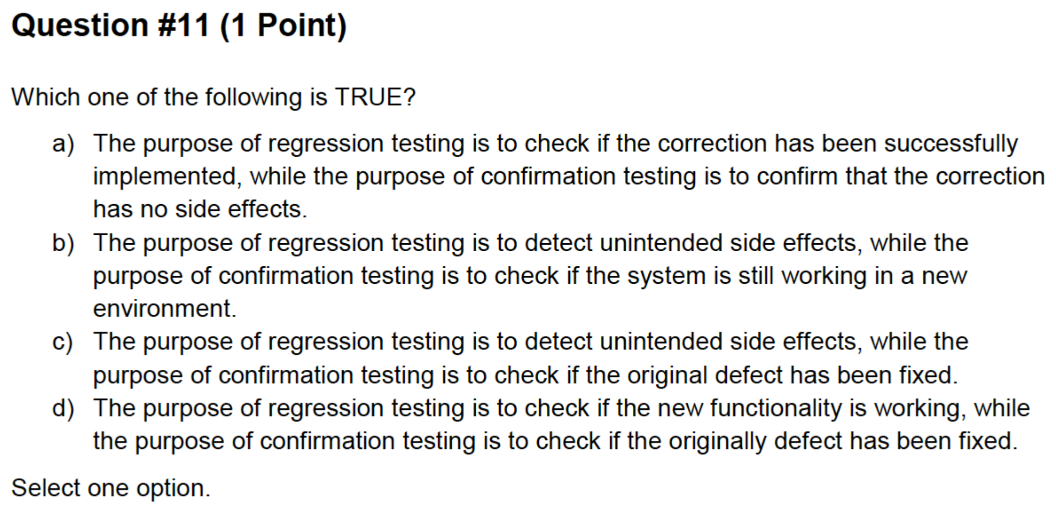
▶ Confirmation testing: After a defect is fixed, the software may be tested with all test cases that failed due to the defect, which should be re-executed on the new software version. The software may also be tested with new tests if, for instance, the defect was missing functionality. At the very least, the steps to reproduce the failure(s) caused by the defect must be re-executed on the new software version. The purpose of a confirmation test is to confirm whether the original defect has been successfully fixed.

▶ Regression testing: It is possible that a change made in one part of the code, whether a fix or another type of change, may accidentally affect the behavior of other parts of the code, whether within the same component, in other components of the same system, or even in other systems.

Changes may include changes to the environment, such as a new version of an operating system or database management system. Such unintended side-effects are called regressions.

Regression testing involves running tests to detect such unintended side-effects.

Confirmation testing and regression testing are performed at all test levels.



Question #11 (1 Point)

Which one of the following is TRUE?

a) The purpose of regression testing is to check if the correction has been successfully implemented, while the purpose of confirmation testing is to confirm that the correction has no side effects.
b) The purpose of regression testing is to detect unintended side effects, while the purpose of confirmation testing is to check if the system is still working in a new environment.
c) The purpose of regression testing is to detect unintended side effects, while the purpose of confirmation testing is to check if the original defect has been fixed.
d) The purpose of regression testing is to check if the new functionality is working, while the purpose of confirmation testing is to check if the originally defect has been fixed.

Select one option.

Especially in iterative and incremental development lifecycles (e.g., Agile), new features, changes to existing features, and code refactoring result in frequent changes to the code, which also requires change-related testing. Due to the evolving nature of the system, confirmation and regression testing are very important. This is particularly relevant for Internet of Things systems where individual objects (e.g., devices) are frequently updated or replaced.

Regression test suites are run many times and generally evolve slowly, so regression testing is a strong candidate for automation. Automation of these tests should start early in the project (see chapter 6).

Maintenance Testing

20- Summarize triggers for maintenance testing (K2)

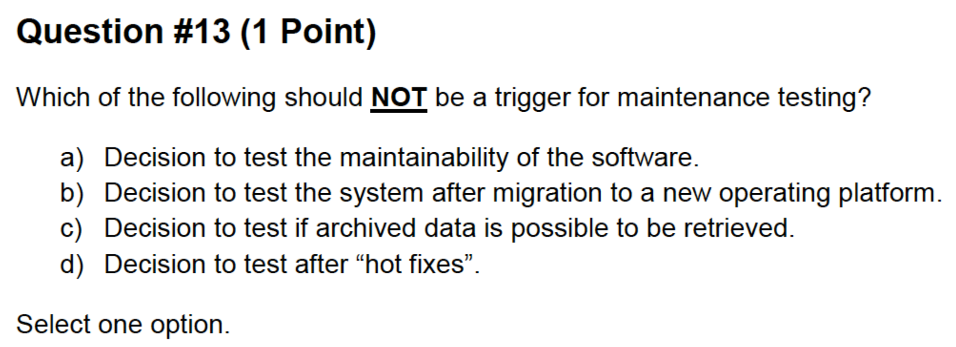
There are several reasons why software maintenance, and thus maintenance testing, takes place, both for planned and unplanned changes.

We can classify the triggers for maintenance as follows:

▶ Modification, such as planned enhancements (e.g., release-based), corrective and emergency changes, changes of the operational environment (such as planned operating system or database upgrades), upgrades of COTS software, and patches for defects and vulnerabilities

▶ Migration, such as from one platform to another, which can require operational tests of the new environment as well as of the changed software, or tests of data conversion when data from another application will be migrated into the system being maintained

▶ Retirement, such as when an application reaches the end of its life When an application or system is retired, this can require testing of data migration or archiving if long data-retention periods are required. Testing restore/retrieve procedures after archiving for long retention periods may also be needed. In addition regression testing may be needed to ensure that any functionality that remains in service still works.



Question #13 (1 Point)

Which of the following should **NOT** be a trigger for maintenance testing?

a) Decision to test the maintainability of the software.
b) Decision to test the system after migration to a new operating platform.
c) Decision to test if archived data is possible to be retrieved.
d) Decision to test after "hot fixes".

Select one option.

For Internet of Things systems, maintenance testing may be triggered by the introduction of completely new or modified things, such as hardware devices and software services, into the overall system. The maintenance testing for such systems places particular emphasis on integration testing at different levels (e.g., network level, application level) and on security aspects, in particular those relating to personal data.

(Shorter and more clear.)

21- Describe the role of impact analysis in maintenance testing (K2)

Impact analysis evaluates the changes that were made for a maintenance release to identify the intended consequences as well as expected and possible side effects of a change, and to identify the areas in the system that will be affected by the change. Impact analysis can also help to identify the impact of a change on existing tests. The side effects and affected areas in the system need to be tested for regressions, possibly after updating any existing tests affected by the change.

Impact analysis may be done before a change is made, to help decide if the change should be made, based on the potential consequences in other areas of the system.

Impact analysis can be difficult if:

- Specifications (e.g., business requirements, user stories, architecture) are out of date or missing
- Test cases are not documented or are out of date
- Bi-directional traceability between tests and the test basis has not been maintained
- Tool support is weak or non-existent
- The people involved do not have domain and/or system knowledge
- Insufficient attention has been paid to the software's maintainability during development

Chapter 3 Static Testing

Removed Contents in Syllabus 2018:

<u>Static Analysis by Tools</u>

- Recall typical defects and errors identified by static analysis and compare them to reviews and dynamic testing
- Describe, using examples, the typical benefits of static analysis
- List typical code and design defects that may be identified by static analysis tools

Removing Reasons:

We want to have more focus on reviews which is more relevant for testers on foundation level.

Static Testing Basics

22- Recognize types of software work product that can be examined by the different static testing techniques (K1)

Almost any work product can be examined using static testing (reviews and/or static analysis), for example:

- ▶ Specifications, including business requirements, functional requirements, and security requirements
- ▶ Epics, user stories, and acceptance criteria
- ▶ Architecture and design specifications
- ▶ Code
- ▶ Testware, including test plans, test cases, test procedures, and automated test scripts
- ▶ User guides
- ▶ Web pages
- ▶ Contracts, project plans, schedules, and budgets
- ▶ Models, such as activity diagrams, which may be used for Model-Based testing (see ISTQB-MBT Foundation Level Model-Based Tester Extension Syllabus and Kramer 2016)

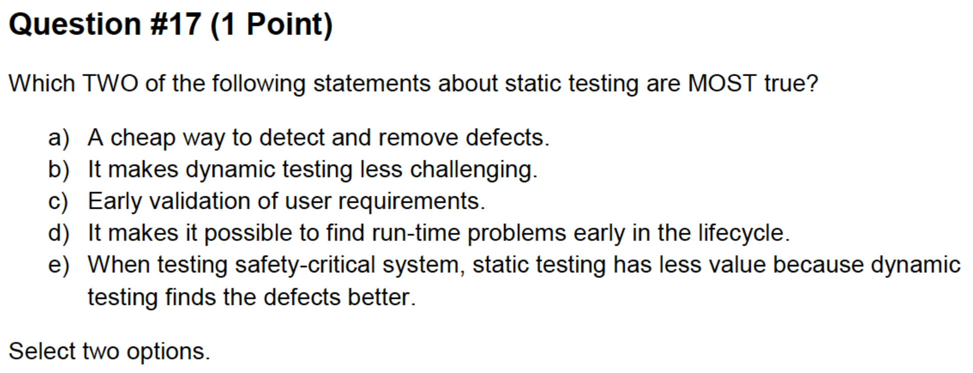
Reviews can be applied to any work product that the participants know how to read and understand. Static analysis can be applied efficiently to any work product with a formal structure (typically code or models) for which an appropriate static analysis tool exists. Static analysis can even be applied with tools that evaluate work products written in natural language such as requirements (e.g., checking for spelling, grammar, and readability).

23- Use examples to describe the value of static testing (K2)

Static testing techniques provide a variety of benefits. When applied early in the software development lifecycle, static testing enables the early detection of defects before dynamic testing is performed (e.g., in requirements or design specifications reviews, product backlog refinement, etc.). Defects found early are often much cheaper to remove than defects found later in the lifecycle, especially compared to defects found after the software is deployed and in active use. Using static testing techniques to find defects and then fixing those defects promptly is almost always much cheaper for the organization than using dynamic testing to find defects in the test object and then fixing them, especially when considering the additional costs associated with updating other work products and performing confirmation and regression testing.

Additional benefits of static testing may include:

- ▶ Detecting and correcting defects more efficiently, and prior to dynamic test execution
- ▶ Identifying defects which are not easily found by dynamic testing
- ▶ Preventing defects in design or coding by uncovering inconsistencies, ambiguities, contradictions, omissions, inaccuracies, and redundancies in requirements
- ▶ Increasing development productivity (e.g., due to improved design, more maintainable code)
- ▶ Reducing development cost and time
- ▶ Reducing testing cost and time
- ▶ Reducing total cost of quality over the software's lifetime, due to fewer failures later in the lifecycle or after delivery into operation
- ▶ Improving communication between team members in the course of participating in reviews



Question #17 (1 Point)

Which TWO of the following statements about static testing are MOST true?

a) A cheap way to detect and remove defects.
b) It makes dynamic testing less challenging.
c) Early validation of user requirements.
d) It makes it possible to find run-time problems early in the lifecycle.
e) When testing safety-critical system, static testing has less value because dynamic testing finds the defects better.

Select two options.

24- Explain the difference between static and dynamic techniques, considering objectives, types of defects to be identified, and the role of these techniques within the software lifecyle (K2)

Static testing and dynamic testing can have the same objectives (see section 1.1.1), such as providing an assessment of the quality of the work products and identifying defects as early as possible. Static and dynamic testing complement each other by finding different types of defects.

One main distinction is that static testing finds defects in work products directly rather than identifying failures caused by defects when the software is run. A defect can reside in a work product for a very long time without causing a failure. The path where the defect lies may be rarely exercised or hard to reach, so it will not be easy to construct and execute a dynamic test that encounters it. Static testing may be able to find the defect with much less effort.

Another distinction is that static testing can be used to improve the consistency and internal quality of work products, while dynamic testing typically focuses on externally visible behaviors.

Compared with dynamic testing, typical defects that are easier and cheaper to find and fix through static testing include:

- Requirement defects (e.g., inconsistencies, ambiguities, contradictions, omissions, inaccuracies, and redundancies)
- Design defects (e.g., inefficient algorithms or database structures, high coupling, low cohesion)
- Coding defects (e.g., variables with undefined values, variables that are declared but never used, unreachable code, duplicate code)
- Deviations from standards (e.g., lack of adherence to coding standards)
- Incorrect interface specifications (e.g., different units of measurement used by the calling system than by the called system)
- Security vulnerabilities (e.g., susceptibility to buffer overflows)
- Gaps or inaccuracies in test basis traceability or coverage (e.g., missing tests for an acceptance criterion)

Moreover, most types of maintainability defects can only be found by static testing (e.g., improper modularization, poor reusability of components, code that is difficult to analyze and modify without introducing new defects).

Review Process

25- Summarize the activities of the work product review process (K2)

The review process comprises the following main activities:

Planning

▶ Defining the scope, which includes the purpose of the review, what documents or parts of documents to review, and the quality characteristics to be evaluated

▶ Estimating effort and timeframe

▶ Identifying review characteristics such as the review type with roles, activities, and checklists

▶ Selecting the people to participate in the review and allocating roles

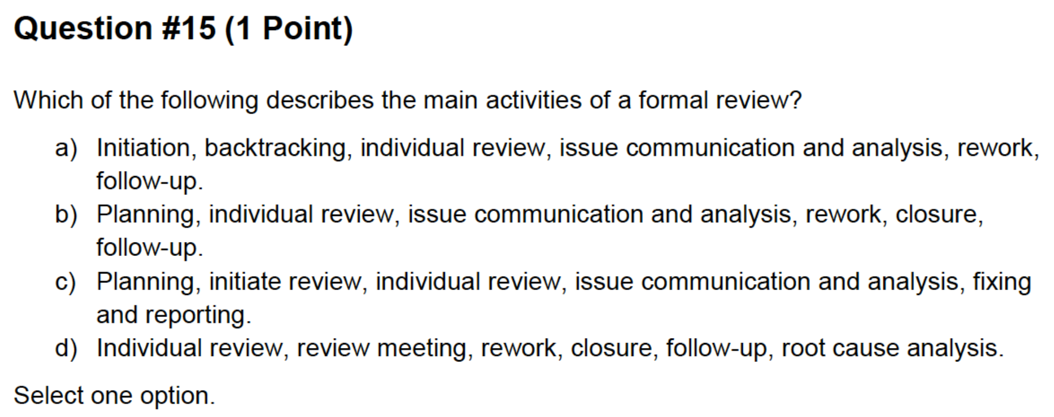
▶ Defining the entry and exit criteria for more formal review types (e.g., inspections)

▶ Checking that entry criteria are met (for more formal review types) Initiate review

▶ Distributing the work product (physically or by electronic means) and other material, such as issue log forms, checklists, and related work products

▶ Explaining the scope, objectives, process, roles, and work products to the participants

▶ Answering any questions that participants may have about the review



Question #15 (1 Point)

Which of the following describes the main activities of a formal review?

a) Initiation, backtracking, individual review, issue communication and analysis, rework, follow-up.
b) Planning, individual review, issue communication and analysis, rework, closure, follow-up.
c) Planning, initiate review, individual review, issue communication and analysis, fixing and reporting.
d) Individual review, review meeting, rework, closure, follow-up, root cause analysis.

Select one option.

Individual review (i.e., individual preparation)

▶ Reviewing all or part of the work product

▶ Noting potential defects, recommendations, and questions

Issue communication and analysis

▶ Communicating identified potential defects (e.g., in a review meeting)

▶ Analyzing potential defects, assigning ownership and status to them

▶ Evaluating and documenting quality characteristics

▶ Evaluating the review findings against the exit criteria to make a review decision (reject; major changes needed; accept, possibly with minor changes)

Fixing and reporting

▶ Creating defect reports for those findings that require changes

▶ Fixing defects found (typically done by the author) in the work product reviewed

▶ Communicating defects to the appropriate person or team (when found in a work product related to the work product reviewed)

▶ Recording updated status of defects (in formal reviews), potentially including the agreement of the comment originator

▶ Gathering metrics (for more formal review types)

▶ Checking that exit criteria are met (for more formal review types)

▶ Accepting the work product when the exit criteria are reached

(As focus on reviews is increased in the 2018 syllabus testers should be able to summarize the activities in their own words)

26- Recognize the different roles and responsibilities in a formal review (K1)

Author

▶ Creates the work product under review

▶ Fixes defects in the work product under review (if necessary)

Management

▶ Is responsible for review planning

▶ Decides on the execution of reviews

▶ Assigns staff, budget, and time

▶ Monitors ongoing cost-effectiveness

▶ Executes control decisions in the event of inadequate outcomes

Facilitator (often called moderator)

▶ Ensures effective running of review meetings (when held)

▶ Mediates, if necessary, between the various points of view

▶ Is often the person upon whom the success of the review depends

Review leader

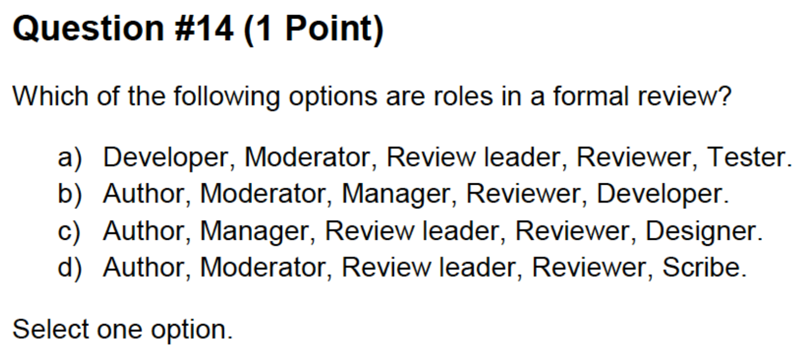
▶ Takes overall responsibility for the review

▶ Decides who will be involved and organizes when and where it will take place

Reviewers

▶ May be subject matter experts, persons working on the project, stakeholders with an interest in the work product, and/or individuals with specific technical or business backgrounds

▶ Identify potential defects in the work product under review



Question #14 (1 Point)

Which of the following options are roles in a formal review?

a) Developer, Moderator, Review leader, Reviewer, Tester.
b) Author, Moderator, Manager, Reviewer, Developer.
c) Author, Manager, Review leader, Reviewer, Designer.
d) Author, Moderator, Review leader, Reviewer, Scribe.

Select one option.

▶ May represent different perspectives (e.g., tester, programmer, user, operator, business analyst, usability expert, etc.)

Scribe (or recorder)

▶ Collates potential defects found during the individual review activity

▶ Records new potential defects, open points, and decisions from the review meeting (when held)

In some review types, one person may play more than one role, and the actions associated with each role may also vary based on review type. In addition, with the advent of tools to support the review process, especially the logging of defects, open points, and decisions, there is often no need for a scribe.

27- Explain the differences between different review types: informal review, walktrough, technical review and inspection (K2)

REVIEW TYPES

	INFORMAL REVIEW	WALKTROUGH	TECHNICAL REVIEW	INSPECTION
Main Purpose	Detecting potential defects.	Find defects, improve the software product, consider alternative implementations, evaluate conformance to standards and specifications.	Gaining consensus, detecting potential defects.	Detecting potential defects, evaluating quality and building confidence in the work product, preventing future similar defects through author learning and root cause analysis.
Possible Additional Purposes	Generating new ideas or solutions, quickly solving minor problems.	Exchanging ideas about techniques or style variations, training of participants, achieving consensus	Evaluating quality and building confidence in the work product, generating new ideas, motivating and enabling authors to improve future work products, considering alternative implementations	Motivating and enabling authors to improve future work products and the software development process, achieving consensus
Process	Not based on a formal (documented) process (May be performed by a colleague of the author (buddy check) or by more people)			
Individual preparation before the review meeting:	(May not involve a review meeting)	Optional	Required	Required
Leading at review meetings	-	Typcally led by the author of the work product	Ideally led by a trained facilitator (typically not the author)	Led by a trained facilitator (not the author)
Reviewers	Varies in usefulness depending on the reviewers		Should be technical peers of the author, and technical experts in the same or other disciplines	Are either peers of the author or experts in other disciplines that are relevant to the work product. AUTHOR CANNOT ACT AS THE REVIEW LEADER, READER OR SCRIBE.
Documentation	Results may be documented	(Scribe)	(Scribe)	(Scribe)
Scribe	-	Mandatory	Mandatory (ideally not the author)	Mandatory
Use of check list	Optional	Optional		
Potential defect logs and review reports	-	May be produced	Typically produced	Are produced
Specified entry and exit criteria	-	-	-	Are used
Other	Very commonly used in Agile development	May take the form of scenarios, dry runs, or simulations		• Uses clearly defined roles, such as those specified in section 3.2.2 which are mandatory and my include a dedicated reader (Who reads the work product aloud during the review meeting) • Metrics are collected and used to improve the entire software development process, including the inspection process.

Question #16 (1 Point)

Which of the review types below is the BEST option to choose when the review must follow a formal process based on rules and checklists?

a) Informal Review.
b) Technical Review.
c) Inspection.
d) Walkthrough.

Select one option.

28- Apply a review technique to a work product to find defects (K3)

There are a number of review techniques that can be applied during the individual review (i.e., individual preparation) activity to uncover defects. These techniques can be used across the review types described above. The effectiveness of the techniques may differ depending on the type of review used. Examples of different individual review techniques for various review types are listed below.

Ad hoc

In an ad hoc review, reviewers are provided with little or no guidance on how this task should be performed. Reviewers often read the work product sequentially, identifying and documenting issues as they encounter them. Ad hoc reviewing is a commonly used technique needing little preparation. This technique is highly dependent on reviewer skills and may lead to many duplicate issues being reported by different reviewers.

Checklist-based

A checklist-based review is a systematic technique, whereby the reviewers detect issues based on checklists that are distributed at review initiation (e.g., by the facilitator). A review checklist consists of a set of questions based on potential defects, which may be derived from experience. Checklists should be specific to the type of work product under review and should be maintained regularly to cover issue types missed in previous reviews. The main advantage of the checklist-based technique is a systematic coverage of typical defect types. Care should be taken not to simply follow the checklist in individual reviewing, but also to look for defects outside the checklist.

Scenarios and dry runs

In a scenario-based review, reviewers are provided with structured guidelines on how to read through the work product. A scenario-based approach supports reviewers in performing “dry runs” on the work product based on expected usage of the work product (if the work product is documented in a suitable format such as use cases). These scenarios provide reviewers with better guidelines on how to identify specific defect types than simple checklist entries. As with checklist-based reviews, in order not to miss other defect types (e.g., missing features), reviewers should not be constrained to the documented scenarios.

Role-based

A role-based review is a technique in which the reviewers evaluate the work product from the perspective of individual stakeholder roles. Typical roles include specific end user types (experienced, inexperienced, senior, child, etc.), and specific roles in the organization (user administrator, system administrator, performance tester, etc.).

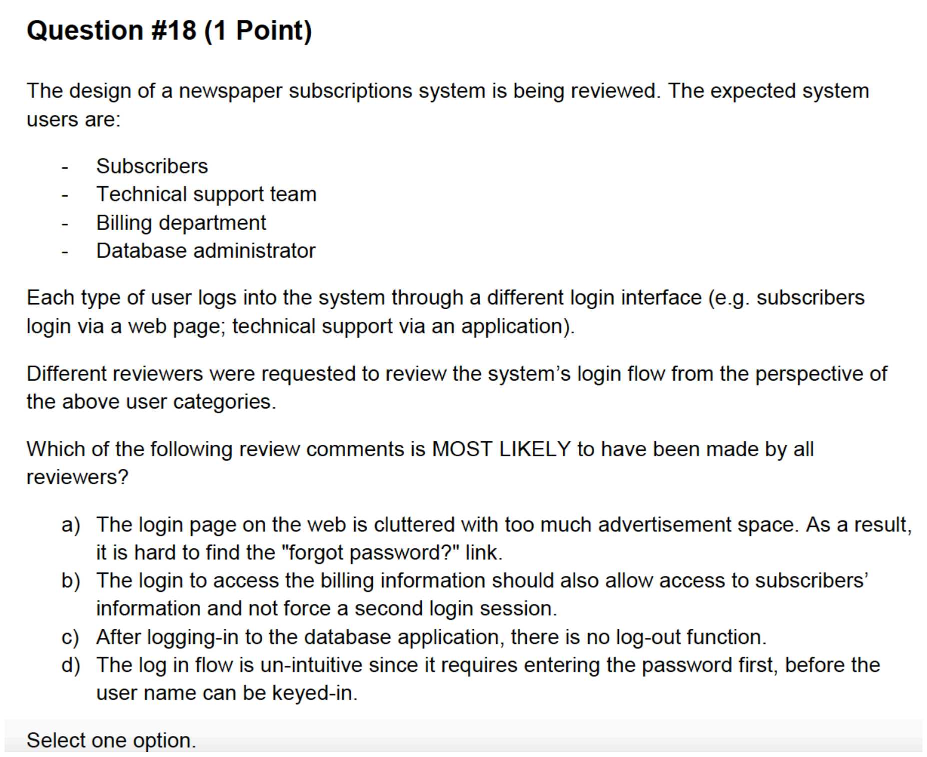
Perspective-based

In perspective-based reading, similar to a role-based review, reviewers take on different stakeholder viewpoints in individual reviewing. Typical stakeholder viewpoints include end user, marketing, designer, tester, or operations. Using different stakeholder viewpoints leads to more depth in individual reviewing with less duplication of issues across reviewers.

In addition, perspective-based reading also requires the reviewers to attempt to use the work product under review to generate the product they would derive from it. For example, a tester would attempt to generate draft acceptance tests if performing a perspective-based reading on a requirements specification to see if all the necessary information was included. Further, in perspective-based reading, checklists are expected to be used.

Empirical studies have shown perspective-based reading to be the most effective general technique for reviewing requirements and technical work products. A key success factor is including and weighing different stakeholder viewpoints appropriately, based on risks. See Shul 2000 for details on perspectivebased reading, and Sauer 2000 for the effectiveness of different review types.

(We want to emphasize static testing and make it more practical within the Foundation syllabus)



Question #18 (1 Point)

The design of a newspaper subscriptions system is being reviewed. The expected system users are:

- Subscribers
- Technical support team
- Billing department
- Database administrator

Each type of user logs into the system through a different login interface (e.g. subscribers login via a web page; technical support via an application).

Different reviewers were requested to review the system’s login flow from the perspective of the above user categories.

Which of the following review comments is MOST LIKELY to have been made by all reviewers?

a) The login page on the web is cluttered with too much advertisement space. As a result, it is hard to find the "forgot password?" link.
b) The login to access the billing information should also allow access to subscribers’ information and not force a second login session.
c) After logging-in to the database application, there is no log-out function.
d) The log in flow is un-intuitive since it requires entering the password first, before the user name can be keyed-in.

Select one option.

29- Explain the factors that contribute to a succesful review (K2)

In order to have a successful review, the appropriate type of review and the techniques used must be considered. In addition, there are a number of other factors that will affect the outcome of the review.

Organizational success factors for reviews include:

- ▶ Each review has clear objectives, defined during review planning, and used as measurable exit criteria
- ▶ Review types are applied which are suitable to achieve the objectives and are appropriate to the type and level of software work products and participants
- ▶ Any review techniques used, such as checklist-based or role-based reviewing, are suitable for effective defect identification in the work product to be reviewed
- ▶ Any checklists used address the main risks and are up to date
- ▶ Large documents are written and reviewed in small chunks, so that quality control is exercised by providing authors early and frequent feedback on defects
- ▶ Participants have adequate time to prepare
- ▶ Reviews are scheduled with adequate notice
- ▶ Management supports the review process (e.g., by incorporating adequate time for review activities in project schedules)

People-related success factors for reviews include:

- ▶ The right people are involved to meet the review objectives, for example, people with different skill sets or perspectives, who may use the document as a work input
- ▶ Testers are seen as valued reviewers who contribute to the review and learn about the work product, which enables them to prepare more effective tests, and to prepare those tests earlier
- ▶ Participants dedicate adequate time and attention to detail
- ▶ Reviews are conducted on small chunks, so that reviewers do not lose concentration during individual review and/or the review meeting (when held)
- ▶ Defects found are acknowledged, appreciated, and handled objectively
- ▶ The meeting is well-managed, so that participants consider it a valuable use of their time
- ▶ The review is conducted in an atmosphere of trust; the outcome will not be used for the evaluation of the participants
- ▶ Participants avoid body language and behaviors that might indicate boredom, exasperation, or hostility to other participants

- ▶ Adequate training is provided, especially for more formal review types such as inspections
- ▶ A culture of learning and process improvement is promoted

(Reworded from "for succesful performance" in 2011 to now "contribute to a succesful review"

Chapter 4 Test Techniques

Removed Contents in Syllabus 2018:

Test Development Process

- Differentiate between a test design specification, test case specification and test procedure specification
- Translate test cases into a well-structured test procedure specification at a level of detail relevant to the knowledge of the testers

Removing Reasons:

Not all terms are often used in a day to day work. We have decided to include the main work products and not include all work products from standards such as IEEE, ISO

Removed to add more time for techniques.

Structure-based or White-box techniques

- Write test cases from given control flows using statement and decision test design techniques.
- Assess statement and decision coverage for completeness with respect to defined exit criteria

Removing Reasons:

Removed after many comments for being too technical for foundation level.

Categories of Test Techniques

30- Explain the characteristics, commonalities, and differences between black-box test techniques, white-box test techniques and experience-based test techniques (K2)

Black-box test techniques (also called behavioral or behavior-based techniques) are based on an analysis of the appropriate test basis (e.g., formal requirements documents, specifications, use cases, user stories, or business processes). These techniques are applicable to both functional and nonfunctional testing. Black-box test techniques concentrate on the inputs and outputs of the test object without reference to its internal structure.

White-box test techniques (also called structural or structure-based techniques) are based on an analysis of the architecture, detailed design, internal structure, or the code of the test object. Unlike black-box test techniques, white-box test techniques concentrate on the structure and processing within the test object.

Experience-based test techniques leverage the experience of developers, testers and users to design, implement, and execute tests. These techniques are often combined with black-box and white-box test techniques.

Common characteristics of black-box test techniques include the following:

▶ Test conditions, test cases, and test data are derived from a test basis that may include software requirements, specifications, use cases, and user stories

▶ Test cases may be used to detect gaps between the requirements and the implementation of the requirements, as well as deviations from the requirements

▶ Coverage is measured based on the items tested in the test basis and the technique applied to the test basis

Common characteristics of white-box test techniques include the following:

▶ Test conditions, test cases, and test data are derived from a test basis that may include code, software architecture, detailed design, or any other source of information regarding the structure of the software

▶ Coverage is measured based on the items tested within a selected structure (e.g., the code or interfaces)

▶ Specifications are often used as an additional source of information to determine the expected outcome of test cases

Common characteristics of experience-based test techniques include the following:

▶ Test conditions, test cases, and test data are derived from a test basis that may include knowledge and experience of testers, developers, users and other stakeholders This knowledge and experience includes expected use of the software, its environment, likely defects, and the distribution of those defects.

Question #20 (1 Point)

Which one of the following options is categorized as a black-box test technique?

a) Techniques based on analysis of the architecture.
b) Techniques checking that the test object is working according to the technical design.
c) Techniques based on the expected use of the software.
d) Techniques based on formal requirements.

Select one option

Black-box Test Techniques

31- Apply equivalence partitioning to derive test cases from given requirements (K3)

Question #25 (1 Point)

An employee's bonus is to be calculated. It cannot be negative, but it can be calculated down to zero. The bonus is based on the length of employment.

The categories are: less than or equal to 2 years, more than 2 years but less than 5 years, 5 or more years, but less than 10 years, 10 years or longer.

What is the minimum number of test cases required to cover all valid equivalence partitions for calculating the bonus?

a) 3.
b) 5.
c) 2.
d) 4.

Select one option.

Question #29 (1 Point)

A video application has the following requirement:
The application shall allow playing a video on the following display sizes:

1. 640x480.
2. 1280x720.
3. 1600x1200.
4. 1920x1080.

Which of the following list of test cases is a result of applying the Equivalence Partitioning test technique to test this requirement?

a) Verify that the application can play a video on a display of size 1920x1080 (1 test).
b) Verify that the application can play a video on a display of size 640x480 and 1920x1080 (2 tests).
c) Verify that the application can play a video on each of the display sizes in the requirement (4 tests).
d) Verify that the application can play a video on any one of the display sizes in the requirement (1 test).

Select one option.

32- Apply boundary value analysis derive test cases from given requirements (K3)

Question #26 (1 Point)

A speed control and reporting system has the following characteristics:
If you drive 50 km/h or less, nothing will happen.
If you drive faster than 50 km/h, but 55 km/h or less, you will be warned.
If you drive faster than 55 km/h but not more than 60 km/h, you will be fined.
If you drive faster than 60 km/h, your driving license will be suspended.

Which would be the most likely set of values (km/h) identified by two-point boundary value analysis?

a) 0, 49, 50, 54, 59, 60.
b) 50, 55, 60.
c) 49, 50, 54, 55, 60, 62.
d) 50, 51, 55, 56, 60, 61.

Select one option.

33- Apply decision table testing derive test cases from given requirements (K3)

Question #27 (1 Point)

A company's employees are paid bonuses if they work more than a year in the company and achieve individually agreed targets.

The following decision table has been designed to test the logic for paying bonuses:

		T1	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**
Conditions									
Cond1	Employment for more than 1 year?	YES	NO	YES	NO	YES	NO	YES	NO
Cond2	Agreed target?	NO	NO	YES	YES	NO	NO	YES	YES
Cond3	Achieved target?	NO	NO	NO	NO	YES	YES	YES	YES
Action									
	Bonus payment?	NO	NO	NO	NO	NO	NO	YES	NO

Which test cases could be eliminated in the above decision table because the test case wouldn't occur in a real situation?

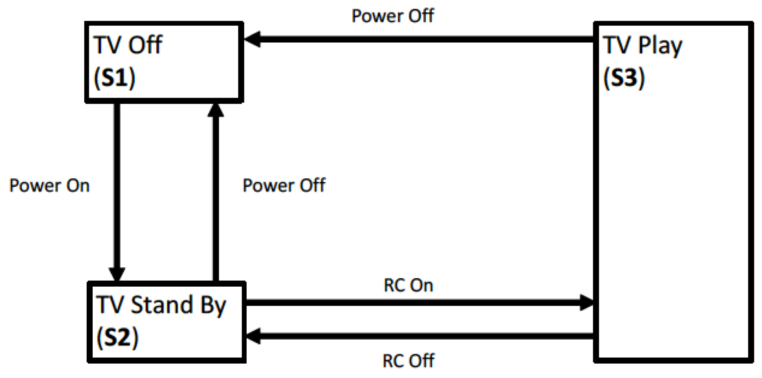
a) T1 and T2.
b) T3 and T4.
c) T7 and T8.
d) T5 and T6.

Select one option.

34- Apply state transition testing derive test cases from given requirements (K3)

Question #28 (1 Point)

Which of the following statements about the given state transition diagram and table of test cases is TRUE?



Test Case	1	2	3	4	5
Start State	S1	S2	S2	S3	S3
Input	Power On	Power Off	RC On	RC Off	Power Off
Expected Final State	S2	S1	S3	S2	S1

a) The given test cases can be used to cover both valid and invalid transitions in the state transition diagram.
b) The given test cases represent all possible valid transitions in the state transition diagram.
c) The given test cases represent only some of the valid transitions in the state transition diagram.
d) The given test cases represent sequential pairs of transitions in the state transition diagram.

Select one option.

35- Explain how to derive test cases from a use case (K2)

Tests can be derived from use cases, which are a specific way of designing interactions with software items, incorporating requirements for the software functions represented by the use cases. Use cases are associated with actors (human users, external hardware, or other components or systems) and subjects (the component or system to which the use case is applied).

Each use case specifies some behavior that a subject can perform in collaboration with one or more actors (UML 2.5.1 2017). A use case can be described by interactions and activities, as well as preconditions, postconditions and natural language where appropriate. Interactions between the actors and the subject may result in changes to the state of the subject. Interactions may be represented graphically by work flows, activity diagrams, or business process models.

A use case can include possible variations of its basic behavior, including exceptional behavior and error handling (system response and recovery from programming, application and communication errors, e.g., resulting in an error message). Tests are designed to exercise the defined behaviors (basic, exceptional or alternative, and error handling). Coverage can be measured by the percentage of use case behaviors tested divided by the total number of use case behaviors, normally expressed as a percentage.

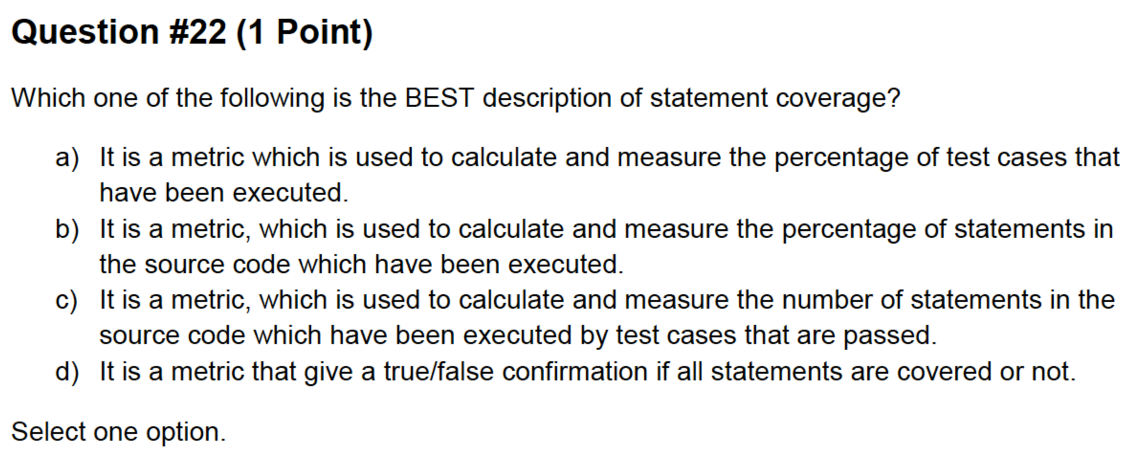
(Focus moved to how Use Cases as a test basis actually are good to derive test cases)

White-box Test Techniques

36- Explain statement coverage (K2)

Statement testing exercises the executable statements in the code. Coverage is measured as the number of statements executed by the tests divided by the total number of executable statements in the test object, normally expressed as a percentage.

(Reworded, simplified and split into two LO)



Question #22 (1 Point)

Which one of the following is the BEST description of statement coverage?

a) It is a metric which is used to calculate and measure the percentage of test cases that have been executed.
b) It is a metric, which is used to calculate and measure the percentage of statements in the source code which have been executed.
c) It is a metric, which is used to calculate and measure the number of statements in the source code which have been executed by test cases that are passed.
d) It is a metric that give a true/false confirmation if all statements are covered or not.

Select one option.

37- Explain decision coverage (K2)

Decision testing exercises the decisions in the code and tests the code that is executed based on the decision outcomes. To do this, the test cases follow the control flows that occur from a decision point (e.g., for an IF statement, one for the true outcome and one for the false outcome; for a CASE statement, test cases would be required for all the possible outcomes, including the default outcome).

Coverage is measured as the number of decision outcomes executed by the tests divided by the total number of decision outcomes in the test object, normally expressed as a percentage.

(Reworded, simplified and split into two LO)

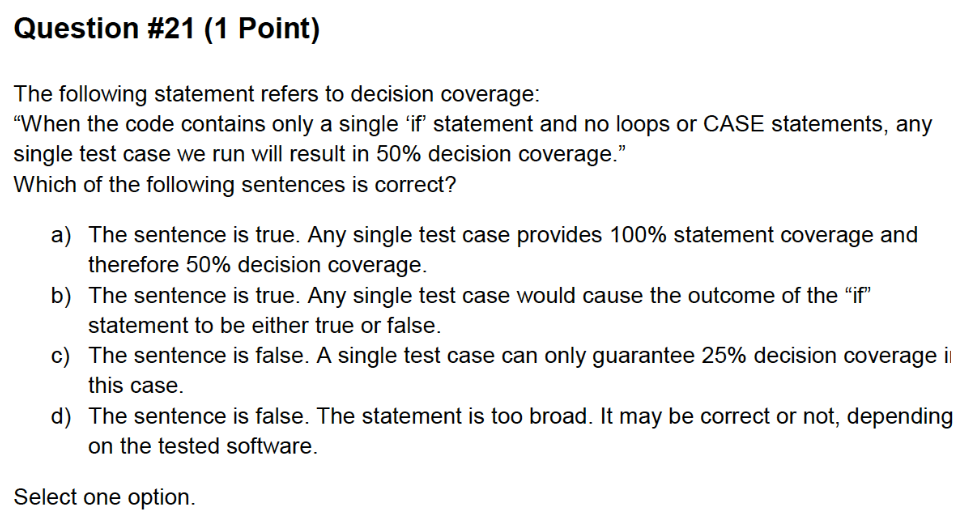
38- Explain the value of statement and decision coverage (K2)

When 100% statement coverage is achieved, it ensures that all executable statements in the code have been tested at least once, but it does not ensure that all decision logic has been tested. Of the two whitebox techniques discussed in this syllabus, statement testing may provide less coverage than decision testing.

When 100% decision coverage is achieved, it executes all decision outcomes, which includes testing the true outcome and also the false outcome, even when there is no explicit false statement (e.g., in the case of an IF statement without an else in the code). Statement coverage helps to find defects in code that was not exercised by other tests. Decision coverage helps to find defects in code where other tests have not taken both true and false outcomes.

Achieving 100% decision coverage guarantees 100% statement coverage (but not vice versa)

(Reworded to focus on value and not the concept)

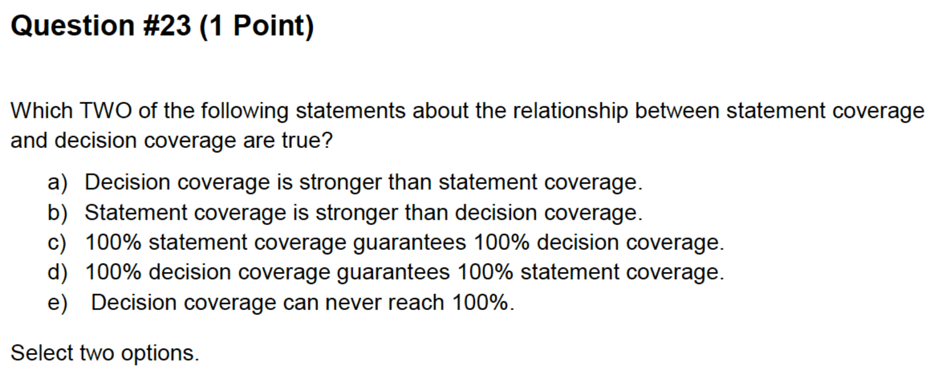


Question #21 (1 Point)

The following statement refers to decision coverage:
“When the code contains only a single ‘if’ statement and no loops or CASE statements, any single test case we run will result in 50% decision coverage.”
Which of the following sentences is correct?

a) The sentence is true. Any single test case provides 100% statement coverage and therefore 50% decision coverage.
b) The sentence is true. Any single test case would cause the outcome of the “if” statement to be either true or false.
c) The sentence is false. A single test case can only guarantee 25% decision coverage i
this case.
d) The sentence is false. The statement is too broad. It may be correct or not, depending on the tested software.

Select one option.



Question #23 (1 Point)

Which TWO of the following statements about the relationship between statement coverage and decision coverage are true?

a) Decision coverage is stronger than statement coverage.
b) Statement coverage is stronger than decision coverage.
c) 100% statement coverage guarantees 100% decision coverage.
d) 100% decision coverage guarantees 100% statement coverage.
e) Decision coverage can never reach 100%.

Select two options.

Experience-based Test Techniques

39- Explain error guessing (K2)

Error guessing is a technique used to anticipate the occurrence of mistakes, defects, and failures, based on the tester's knowledge, including:

- ▶ How the application has worked in the past
- ▶ What types of mistakes the developers tend to make
- ▶ Failures that have occurred in other applications

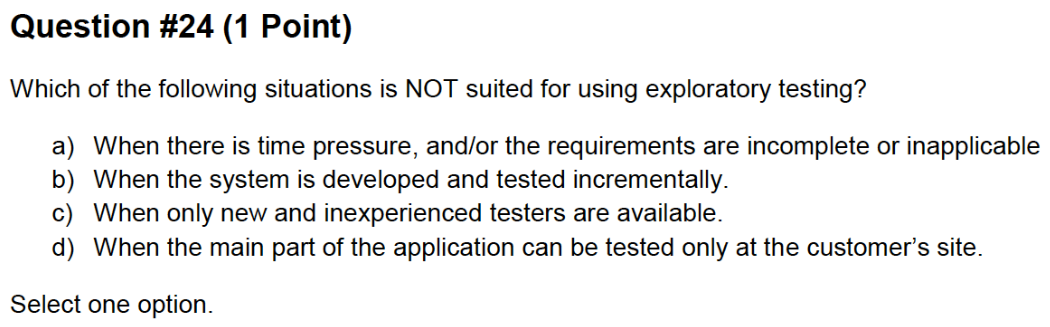
A methodical approach to the error guessing technique is to create a list of possible mistakes, defects, and failures, and design tests that will expose those failures and the defects that caused them. These mistake, defect, failure lists can be built based on experience, defect and failure data, or from common knowledge about why software fails.

(Included because this is a commonly used technique (from ISTQB survey.))

40- Explain exploratory testing (K2)

In exploratory testing, informal (not pre-defined) tests are designed, executed, logged, and evaluated dynamically during test execution. The test results are used to learn more about the component or system, and to create tests for the areas that may need more testing.

Exploratory testing is sometimes conducted using session-based testing to structure the activity. In session-based testing, exploratory testing is conducted within a defined time-box, and the tester uses a test charter containing test objectives to guide the testing. The tester may use test session sheets to document the steps followed and the discoveries made.



Question #24 (1 Point)

Which of the following situations is NOT suited for using exploratory testing?

a) When there is time pressure, and/or the requirements are incomplete or inapplicable
b) When the system is developed and tested incrementally.
c) When only new and inexperienced testers are available.
d) When the main part of the application can be tested only at the customer's site.

Select one option.

Exploratory testing is most useful when there are few or inadequate specifications or significant time pressure on testing. Exploratory testing is also useful to complement other more formal testing techniques.

Exploratory testing is strongly associated with reactive test strategies (see section 5.2.2). Exploratory testing can incorporate the use of other black-box, white-box, and experience-based techniques.

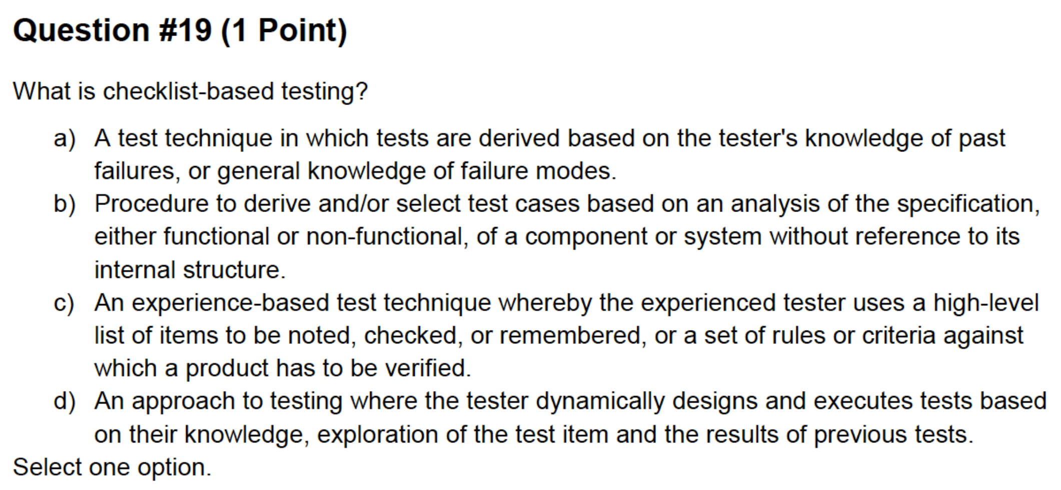
(Simplified and focus only on exploratory testing, a commonly used technique (from ISTQB survey))

41- Explain checklist-based testing (K2)

In checklist-based testing, testers design, implement, and execute tests to cover test conditions found in a checklist. As part of analysis, testers create a new checklist or expand an existing checklist, but testers may also use an existing checklist without modification. Such checklists can be built based on experience, knowledge about what is important for the user, or an understanding of why and how software fails.

Checklists can be created to support various test types, including functional and non-functional testing. In the absence of detailed test cases, checklist-based testing can provide guidelines and a degree of consistency. As these are high-level lists, some variability in the actual testing is likely to occur, resulting in potentially greater coverage but less repeatability.

(Included because this is a commonly used technique (from ISTQB survey.))



Question #19 (1 Point)

What is checklist-based testing?

a) A test technique in which tests are derived based on the tester's knowledge of past failures, or general knowledge of failure modes.
b) Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.
c) An experience-based test technique whereby the experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified.
d) An approach to testing where the tester dynamically designs and executes tests based on their knowledge, exploration of the test item and the results of previous tests.

Select one option.

Chapter 5 Test Management

Test Organization

42- Explain the benefits and drawbacks of independent testing (K2)

Testing tasks may be done by people in a specific testing role, or by people in another role (e.g., customers). A certain degree of independence often makes the tester more effective at finding defects due to differences between the author's and the tester's cognitive biases (see section 1.5). Independence is not, however, a replacement for familiarity, and developers can efficiently find many defects in their own code.

Degrees of independence in testing include the following (from low level of independence to high level):

- ▶ No independent testers; the only form of testing available is developers testing their own code
- ▶ Independent developers or testers within the development teams or the project team; this could be developers testing their colleagues' products
- ▶ Independent test team or group within the organization, reporting to project management or executive management
- ▶ Independent testers from the business organization or user community, or with specializations in specific test types such as usability, security, performance, regulatory/compliance, or portability
- ▶ Independent testers external to the organization, either working on-site (insourcing) or off-site (outsourcing)

For most types of projects, it is usually best to have multiple test levels, with some of these levels handled by independent testers. Developers should participate in testing, especially at the lower levels, so as to exercise control over the quality of their own work.

The way in which independence of testing is implemented varies depending on the software development lifecycle model. For example, in Agile development, testers may be part of a development team. In some organizations using Agile methods, these testers may be considered part of a larger independent test team as well. In addition, in such organizations, product owners may perform acceptance testing to validate user stories at the end of each iteration.

Potential benefits of test independence include:

- ▶ Independent testers are likely to recognize different kinds of failures compared to developers because of their different backgrounds, technical perspectives, and biases

- An independent tester can verify, challenge, or disprove assumptions made by stakeholders during specification and implementation of the system

Potential drawbacks of test independence include:

- Isolation from the development team, leading to a lack of collaboration, delays in providing feedback to the development team, or an adversarial relationship with the development team
- Developers may lose a sense of responsibility for quality
- Independent testers may be seen as a bottleneck or blamed for delays in release
- Independent testers may lack some important information (e.g., about the test object)

Many organizations are able to successfully achieve the benefits of test independence while avoiding the drawbacks.

(Simplified and reduced from 2011)

43- Identify the tasks of a Test Manager and Tester (K1)

In this syllabus, two test roles are covered, test managers and testers. The activities and tasks performed by these two roles depend on the project and product context, the skills of the people in the roles, and the organization.

The test manager is tasked with overall responsibility for the test process and successful leadership of the test activities. The test management role might be performed by a professional test manager, or by a project manager, a development manager, or a quality assurance manager. In larger projects or organizations, several test teams may report to a test manager, test coach, or test coordinator, each team being headed by a test leader or lead tester.

Typical test manager tasks may include:

- Develop or review a test policy and test strategy for the organization
- Plan the test activities by considering the context, and understanding the test objectives and risks. This may include selecting test approaches, estimating test time, effort and cost, acquiring resources, defining test levels and test cycles, and planning defect management
- Write and update the test plan(s)
- Coordinate the test plan(s) with project managers, product owners, and others
- Share testing perspectives with other project activities, such as integration planning
- Initiate the analysis, design, implementation, and execution of tests, monitor test progress and results, and check the status of exit criteria (or definition of done)
- Prepare and deliver test progress reports and test summary reports based on the information gathered

- Adapt planning based on test results and progress (sometimes documented in test progress reports, and/or in test summary reports for other testing already completed on the project) and take any actions necessary for test control
- Support setting up the defect management system and adequate configuration management of testware
- Introduce suitable metrics for measuring test progress and evaluating the quality of the testing and the product
- Support the selection and implementation of tools to support the test process, including recommending the budget for tool selection (and possibly purchase and/or support), allocating time and effort for pilot projects, and providing continuing support in the use of the tool(s)
- Decide about the implementation of test environment(s)
- Promote and advocate the testers, the test team, and the test profession within the organization
- Develop the skills and careers of testers (e.g., through training plans, performance evaluations, coaching, etc.)

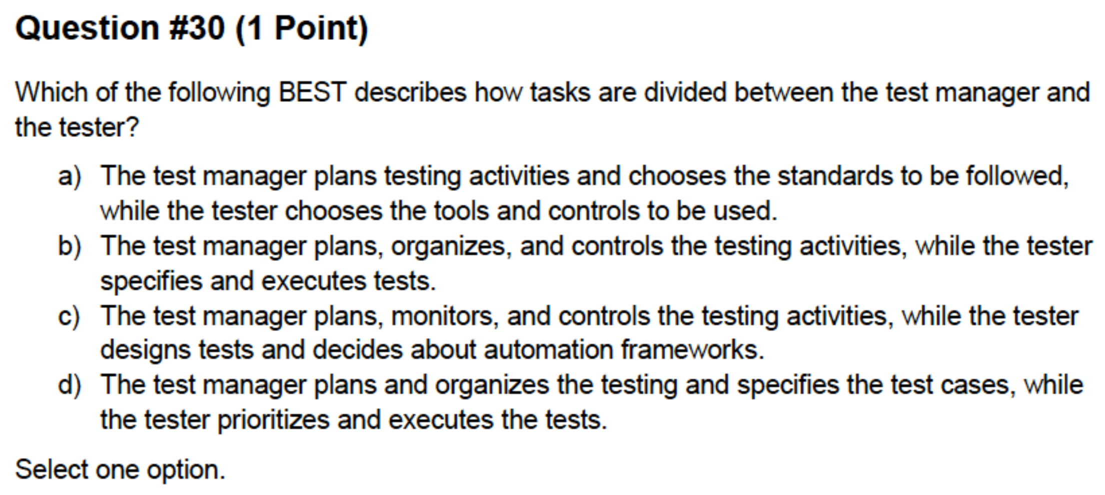
The way in which the test manager role is carried out varies depending on the software development lifecycle. For example, in Agile development, some of the tasks mentioned above are handled by the Agile team, especially those tasks concerned with the day-to-day testing done within the team, often by a tester working within the team. Some of the tasks that span multiple teams or the entire organization, or that have to do with personnel management, may be done by test managers outside of the development team, who are sometimes called test coaches. See Black 2009 for more on managing the test process.

Typical tester tasks may include:

- Review and contribute to test plans
- Analyze, review, and assess requirements, user stories and acceptance criteria, specifications, and models for testability (i.e., the test basis)
- Identify and document test conditions, and capture traceability between test cases, test conditions, and the test basis
- Design, set up, and verify test environment(s), often coordinating with system administration and network management
- Design and implement test cases and test procedures
- Prepare and acquire test data
- Create the detailed test execution schedule
- Execute tests, evaluate the results, and document deviations from expected results
- Use appropriate tools to facilitate the test process
- Automate tests as needed (may be supported by a developer or a test automation expert)
- Evaluate non-functional characteristics such as performance efficiency, reliability, usability, security, compatibility, and portability

- Review tests developed by others

People who work on test analysis, test design, specific test types, or test automation may be specialists in these roles. Depending on the risks related to the product and the project, and the software development lifecycle model selected, different people may take over the role of tester at different test levels. For example, at the component testing level and the component integration testing level, the role of a tester is often done by developers. At the acceptance test level, the role of a tester is often done by business analysts, subject matter experts, and users. At the system test level and the system integration test level, the role of a tester is often done by an independent test team. At the operational acceptance test level, the role of a tester is often done by operations and/or systems administration staff.



Question #30 (1 Point)

Which of the following BEST describes how tasks are divided between the test manager and the tester?

a) The test manager plans testing activities and chooses the standards to be followed, while the tester chooses the tools and controls to be used.
b) The test manager plans, organizes, and controls the testing activities, while the tester specifies and executes tests.
c) The test manager plans, monitors, and controls the testing activities, while the tester designs tests and decides about automation frameworks.
d) The test manager plans and organizes the testing and specifies the test cases, while the tester prioritizes and executes the tests.

Select one option.

Test Planning and Estimation

44- Summarize the purpose and content of a test plan (K2)

A test plan outlines test activities for development and maintenance projects. Planning is influenced by the test policy and test strategy of the organization, the development lifecycles and methods being used (see section 2.1), the scope of testing, objectives, risks, constraints, criticality, testability, and the availability of resources.

As the project and test planning progress, more information becomes available and more detail can be included in the test plan. Test planning is a continuous activity and is performed throughout the product's lifecycle. (Note that the product's lifecycle may extend beyond a project's scope to include the maintenance phase.) Feedback from test activities should be used to recognize changing risks so that planning can be adjusted. Planning may be documented in a master test plan and in separate test plans for test levels, such as system testing and acceptance testing, or for separate test types, such as usability testing and performance testing. Test planning activities may include the following and some of these may be documented in a test plan:

▶ Determining the scope, objectives, and risks of testing

▶ Defining the overall approach of testing

▶ Integrating and coordinating the test activities into the software lifecycle activities

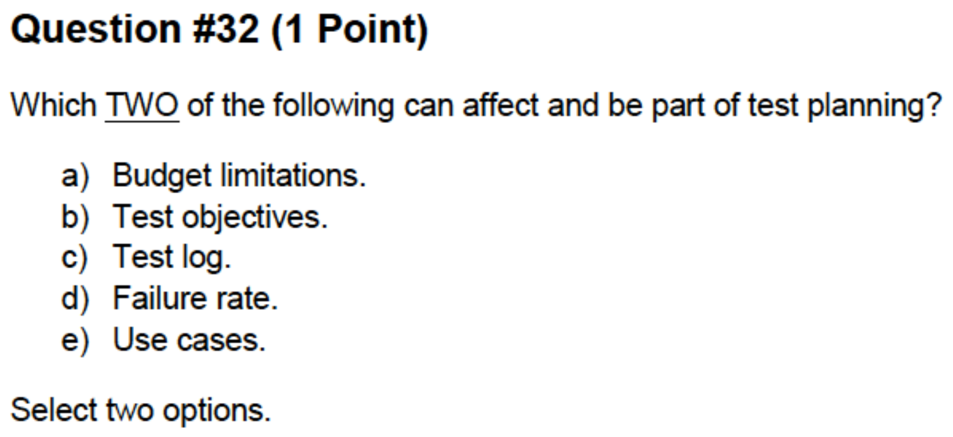
▶ Making decisions about what to test, the people and other resources required to perform the various test activities, and how test activities will be carried out

▶ Scheduling of test analysis, design, implementation, execution, and evaluation activities, either on particular dates (e.g., in sequential development) or in the context of each iteration (e.g., in iterative development)

▶ Selecting metrics for test monitoring and control

▶ Budgeting for the test activities

▶ Determining the level of detail and structure for test documentation (e.g., by providing templates or example documents)



Question #32 (1 Point)

Which TWO of the following can affect and be part of test planning?

a) Budget limitations.
b) Test objectives.
c) Test log.
d) Failure rate.
e) Use cases.

Select two options.

45- Differentiate between various test approaches (K2)

A test strategy provides a generalized description of the test process, usually at the product or organizational level. Common types of test strategies include:

▶ **Analytical**: This type of test strategy is based on an analysis of some factor (e.g., requirement or risk). Risk-based testing is an example of an analytical approach, where tests are designed and prioritized based on the level of risk.

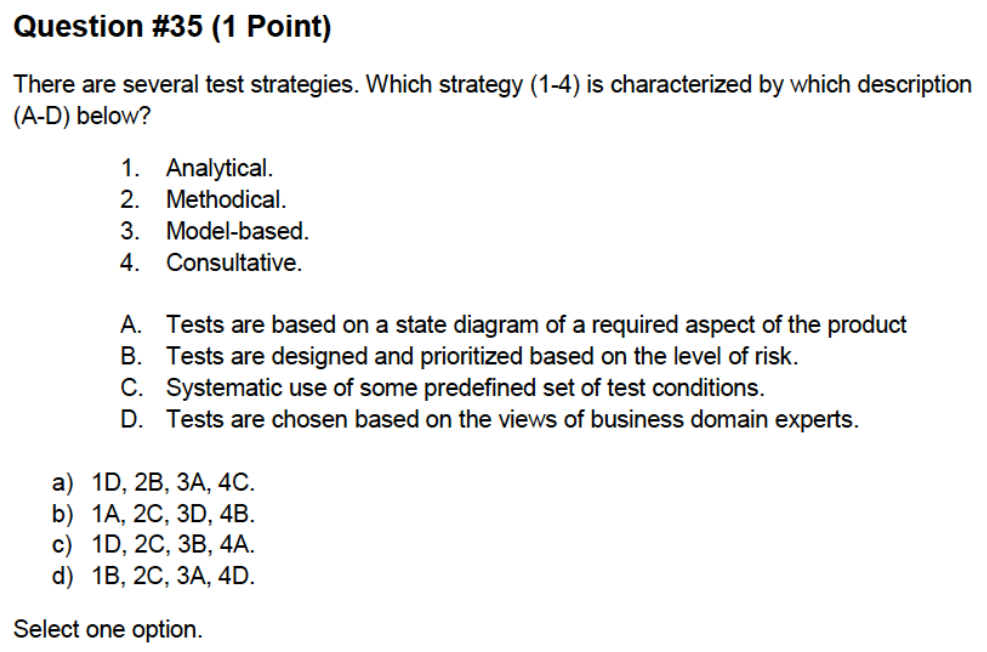
▶ **Model-Based:** In this type of test strategy, tests are designed based on some model of some required aspect of the product, such as a function, a business process, an internal structure, or a non-functional characteristic (e.g., reliability). Examples of such models include business process models, state models, and reliability growth models.

▶ **Methodical**: This type of test strategy relies on making systematic use of some predefined set of tests or test conditions, such as a taxonomy of common or likely types of failures, a list of important quality characteristics, or company-wide look-and-feel standards for mobile apps or web pages.

▶ **Process-compliant (or standard-compliant):** This type of test strategy involves analyzing, designing, and implementing tests based on external rules and standards, such as those specified by industry-specific standards, by process documentation, by the rigorous identification and use of the test basis, or by any process or standard imposed on or by the organization.

▶ **Directed (or consultative):** This type of test strategy is driven primarily by the advice, guidance, or instructions of stakeholders, business domain experts, or technology experts, who may be outside the test team or outside the organization itself.

▶ **Regression-averse**: This type of test strategy is motivated by a desire to avoid regression of existing capabilities. This test strategy includes reuse of existing testware (especially test cases and test data), extensive automation of regression tests, and standard test suites.



Question #35 (1 Point)

There are several test strategies. Which strategy (1-4) is characterized by which description (A-D) below?

1. Analytical.
2. Methodical.
3. Model-based.
4. Consultative.

A. Tests are based on a state diagram of a required aspect of the product
B. Tests are designed and prioritized based on the level of risk.
C. Systematic use of some predefined set of test conditions.
D. Tests are chosen based on the views of business domain experts.

a) 1D, 2B, 3A, 4C.
b) 1A, 2C, 3D, 4B.
c) 1D, 2C, 3B, 4A.
d) 1B, 2C, 3A, 4D.

Select one option.

▶ **Reactive**: In this type of test strategy, testing is reactive to the component or system being tested, and the events occurring during test execution, rather than being pre-planned (as the preceding strategies are). Tests are designed and implemented, and may immediately be executed in response to knowledge gained from prior test results. Exploratory testing is a common technique employed in reactive strategies.

An appropriate test strategy is often created by combining several of these types of test strategies. For example, risk-based testing (an analytical strategy) can be combined with exploratory testing (a reactive strategy); they complement each other and may achieve more effective testing when used together.

While the test strategy provides a generalized description of the test process, the test approach tailors the test strategy for a particular project or release. The test approach is the starting point for selecting the test techniques, test levels, and test types, and for defining the entry criteria and exit criteria (or definition of ready and definition of done, respectively). The tailoring of the strategy is based on decisions made in relation to the complexity and goals of the project, the type of product being developed, and product risk analysis. The selected approach depends on the context and may consider factors such as risks, safety, available resources and skills, technology, the nature of the system (e.g., custom-built versus COTS), test objectives, and regulations.

(Simplified as this is more advanced knowledge for a Test Manager, compared to what we think you need to know as a Foundation Certified Tester.)

46- Give examples of potential entry and exit criteria (K2)

Typical entry criteria include:

▶ Availability of testable requirements, user stories, and/or models (e.g., when following a modelbased testing strategy)

▶ Availability of test items that have met the exit criteria for any prior test levels

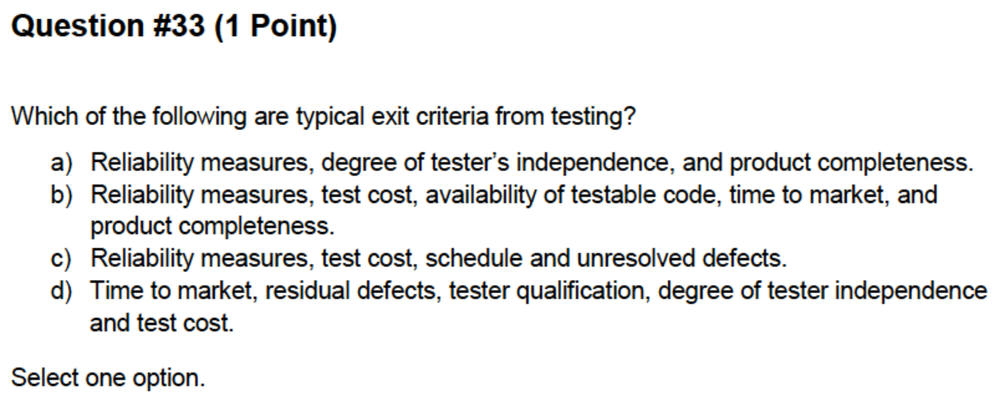
▶ Availability of test environment

▶ Availability of necessary test tools

▶ Availability of test data and other necessary resources

Typical exit criteria include:

▶ Planned tests have been executed



Question #33 (1 Point)

Which of the following are typical exit criteria from testing?

a) Reliability measures, degree of tester's independence, and product completeness.
b) Reliability measures, test cost, availability of testable code, time to market, and product completeness.
c) Reliability measures, test cost, schedule and unresolved defects.
d) Time to market, residual defects, tester qualification, degree of tester independence and test cost.

Select one option.

- A defined level of coverage (e.g., of requirements, user stories, acceptance criteria, risks, code) has been achieved
- The number of unresolved defects is within an agreed limit
- The number of estimated remaining defects is sufficiently low
- The evaluated levels of reliability, performance efficiency, usability, security, and other relevant quality characteristics are sufficient

Even without exit criteria being satisfied, it is also common for test activities to be curtailed due to the budget being expended, the scheduled time being completed, and/or pressure to bring the product to market. It can be acceptable to end testing under such circumstances, if the project stakeholders and business owners have reviewed and accepted the risk to go live without further testing.

(*Simplified*)

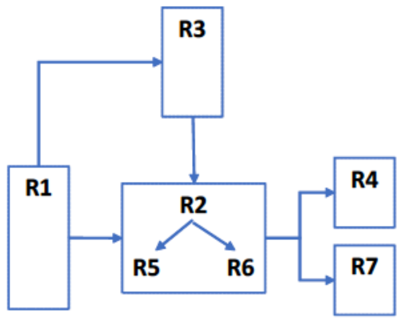
47- Apply knowledge of prioritization, and technical and logical dependencies, to schedule test execution for a given set of test cases (K3)

Once the various test cases and test procedures are produced (with some test procedures potentially automated) and assembled into test suites, the test suites can be arranged in a test execution schedule that defines the order in which they are to be run. The test execution schedule should take into account such factors as prioritization, dependencies, confirmation tests, regression tests, and the most efficient sequence for executing the tests.

Ideally, test cases would be ordered to run based on their priority levels, usually by executing the test cases with the highest priority first. However, this practice may not work if the test cases have dependencies or the features being tested have dependencies. If a test case with a higher priority is dependent on a test case with a

Question #37 (1 Point)

The following diagram shows the logical dependencies between a set of seven requirements, where a dependency is shown by an arrow. For example, "R1 -> R3" means that R3 depends on R1.



Which one of the following options structures the test execution schedule according to the requirement dependencies?

a) R1 → R3 → R1 → R2 → R5 → R6 → R4 → R7.
b) R1 → R3 → R2 → R5 → R2 → R6 → R4 → R7.
c) R1 → R3 → R2 → R5 → R6 → R4 → R7.
d) R1 → R2 → R5 → R6 → R3 → R4 → R7.

Select one option.

lower priority, the lower priority test case must be executed first.

Similarly, if there are dependencies across test cases, they must be ordered appropriately regardless of their relative priorities. Confirmation and regression tests must be prioritized as well, based on the importance of rapid feedback on changes, but here again dependencies may apply.

In some cases, various sequences of tests are possible, with differing levels of efficiency associated with those sequences. In such cases, trade-offs between efficiency of test execution versus adherence to prioritization must be made.

48- Identify factors that influence the effort related to testing (K1)

Test effort estimation involves predicting the amount of test-related work that will be needed in order to meet the objectives of the testing for a particular project, release, or iteration. Factors influencing the test effort may include characteristics of the product, characteristics of the development process, characteristics of the people, and the test results, as shown below.

Product characteristics

- ▶ The risks associated with the product
- ▶ The quality of the test basis
- ▶ The size of the product
- ▶ The complexity of the product domain
- ▶ The requirements for quality characteristics (e.g., security, reliability)
- ▶ The required level of detail for test documentation
- ▶ Requirements for legal and regulatory compliance

Development process characteristics

- ▶ The stability and maturity of the organization
- ▶ The development model in use
- ▶ The test approach
- ▶ The tools used
- ▶ The test process
- ▶ Time pressure

People characteristics

- ▶ The skills and experience of the people involved, especially with similar projects and products (e.g., domain knowledge)
- ▶ Team cohesion and leadership

Test results

- ▶ The number and severity of defects found
- ▶ The amount of rework required

49- Explain the difference between two estimation techniques: the metrics-based technique and the expert-based technique (K2)

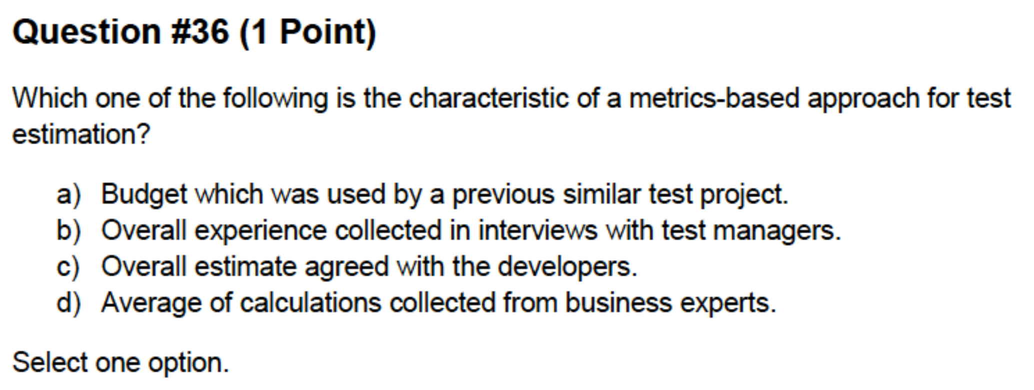
There are a number of estimation techniques used to determine the effort required for adequate testing.

Two of the most commonly used techniques are:

- ▶ The metrics-based technique: estimating the test effort based on metrics of former similar projects, or based on typical values
- ▶ The expert-based technique: estimating the test effort based on the experience of the owners of the testing tasks or by experts

For example, in Agile development, burndown charts are examples of the metrics-based approach as effort is being captured and reported, and is then used to feed into the team's velocity to determine the amount of work the team can do in the next iteration; whereas planning poker is an example of the expert-based approach, as team members are estimating the effort to deliver a feature based on their experience (ISTQB-AT Foundation Level Agile Tester Extension Syllabus).

Within sequential projects, defect removal models are examples of the metrics-based approach, where volumes of defects and time to remove them are captured and reported, which then provides a basis for estimating future projects of a similar nature; whereas the Wideband Delphi estimation technique is an example of the expert-based approach in which groups of experts provides estimates based on their experience



Question #36 (1 Point)

Which one of the following is the characteristic of a metrics-based approach for test estimation?

a) Budget which was used by a previous similar test project.
b) Overall experience collected in interviews with test managers.
c) Overall estimate agreed with the developers.
d) Average of calculations collected from business experts.

Select one option.

Test Monitoring and Control

50- Recall metrics used for testing (K1)

Metrics can be collected during and at the end of test activities in order to assess:

▶ Progress against the planned schedule and budget

▶ Current quality of the test object

▶ Adequacy of the test approach

▶ Effectiveness of the test activities with respect to the objectives

Common test metrics include:

▶ Percentage of planned work done in test case preparation (or percentage of planned test cases implemented)

▶ Percentage of planned work done in test environment preparation

▶ Test case execution (e.g., number of test cases run/not run, test cases passed/failed, and/or test conditions passed/failed)

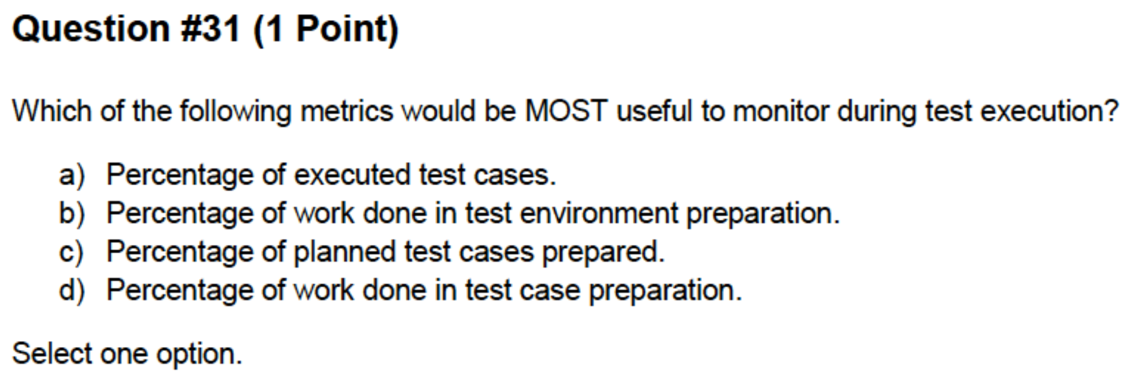
▶ Defect information (e.g., defect density, defects found and fixed, failure rate, and confirmation test results)

▶ Test coverage of requirements, user stories, acceptance criteria, risks, or code

▶ Task completion, resource allocation and usage, and effort

▶ Cost of testing, including the cost compared to the benefit of finding the next defect or the cost compared to the benefit of running the next test

(Expanded to include all activities of testing, not just test preparation and execution)



Question #31 (1 Point)

Which of the following metrics would be MOST useful to monitor during test execution?

a) Percentage of executed test cases.
b) Percentage of work done in test environment preparation.
c) Percentage of planned test cases prepared.
d) Percentage of work done in test case preparation.

Select one option.

51- Summarize the purposes, contents and audiences for test reports (K2)

The purpose of test reporting is to summarize and communicate test activity information, both during and at the end of a test activity (e.g., a test level). The test report prepared during a test activity may be referred to as a test progress report, while a test report prepared at the end of a test activity may be referred to as a test summary report.

During test monitoring and control, the test manager regularly issues test progress reports for stakeholders. In addition to content common to test progress reports and test summary reports, typical

test progress reports may also include:

- The status of the test activities and progress against the test plan
- Factors impeding progress
- Testing planned for the next reporting period
- The quality of the test object

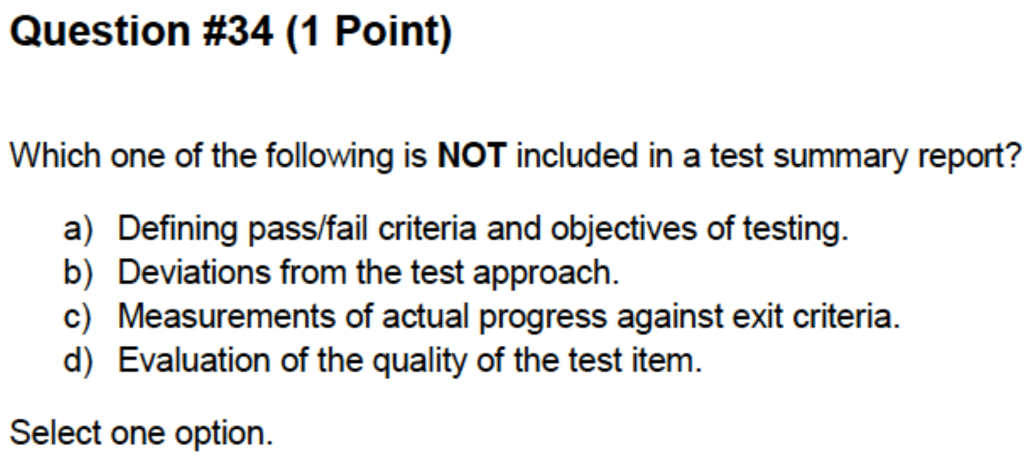
When exit criteria are reached, the test manager issues the test summary report. This report provides a summary of the testing performed, based on the latest test progress report and any other relevant information.

Typical test progress reports and test summary reports may include:

- Summary of testing performed
- Information on what occurred during a test period
- Deviations from plan, including deviations in schedule, duration, or effort of test activities
- Status of testing and product quality with respect to the exit criteria or definition of done
- Factors that have blocked or continue to block progress
- Metrics of defects, test cases, test coverage, activity progress, and resource consumption. (e.g., as described in 5.3.1)
- Residual risks (see section 5.5)
- Reusable test work products produced

The contents of a test report will vary depending on the project, the organizational requirements, and the software development lifecycle. For example, a complex project with many stakeholders or a regulated project may require more detailed and rigorous reporting than a quick software update. As another example, in Agile development, test progress reporting may be incorporated into task boards, defect summaries, and burndown charts, which may be discussed during a daily stand-up meeting (see ISTQBAT Foundation Level Agile Tester Extension Syllabus).

In addition to tailoring test reports based on the context of the project, test reports should be tailored based on the report's audience. The type and amount of information that should be included for a technical audience or a test team may be different from what would be included in an executive summary report. In the first case, detailed information on defect types and trends may be important. In the latter case, a high-level report (e.g., a status summary of defects by priority, budget, schedule, and test conditions passed/failed/not tested) may be more appropriate. ISO standard (ISO/IEC/IEEE 29119-3) refers to two types of test reports, test progress reports and test completion reports (called test summary reports in this syllabus), and contains structures and examples for each type.



Question #34 (1 Point)

Which one of the following is **NOT** included in a test summary report?

a) Defining pass/fail criteria and objectives of testing.
b) Deviations from the test approach.
c) Measurements of actual progress against exit criteria.
d) Evaluation of the quality of the test item.

Select one option.

Configuration Management

52- Summarize how configuration management supports testing (K2)

The purpose of configuration management is to establish and maintain the integrity of the component or system, the testware, and their relationships to one another through the project and product lifecycle.

To properly support testing, configuration management may involve ensuring the following:

- All test items are uniquely identified, version controlled, tracked for changes, and related to each other
- All items of testware are uniquely identified, version controlled, tracked for changes, related to each other and related to versions of the test item(s) so that traceability can be maintained throughout the test process

- All identified documents and software items are referenced unambiguously in test documentation

During test planning, configuration management procedures and infrastructure (tools) should be identified and implemented.

Risks and Testing

53- Define risk level by using likelihood and impact (K1)

Risk involves the possibility of an event in the future which has negative consequences. The level of risk is determined by the likelihood of the event and the impact (the harm) from that event.

54- Distinguish between project and product risks (K2)

Product risk involves the possibility that a work product (e.g., a specification, component, system, or test) may fail to satisfy the legitimate needs of its users and/or stakeholders. When the product risks are associated with specific quality characteristics of a product (e.g., functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability, and portability), product risks are also called quality risks. Examples of product risks include:

- Software might not perform its intended functions according to the specification
- Software might not perform its intended functions according to user, customer, and/or stakeholder needs
- A system architecture may not adequately support some non-functional requirement(s)
- A particular computation may be performed incorrectly in some circumstances
- A loop control structure may be coded incorrectly
- Response-times may be inadequate for a high-performance transaction processing system
- User experience (UX) feedback might not meet product expectations

Project risk involves situations that, should they occur, may have a negative effect on a project's ability to achieve its objectives. Examples of project risks include:

- **Project issues:**
    - o Delays may occur in delivery, task completion, or satisfaction of exit criteria or definition of done
    - o Inaccurate estimates, reallocation of funds to higher priority projects, or general costcutting across the organization may result in inadequate funding
    - o Late changes may result in substantial re-work

- **Organizational issues:**
  - o Skills, training, and staff may not be sufficient
  - o Personnel issues may cause conflict and problems
  - o Users, business staff, or subject matter experts may not be available due to conflicting business priorities
- **Political issues:**
  - o Testers may not communicate their needs and/or the test results adequately
  - o Developers and/or testers may fail to follow up on information found in testing and reviews (e.g., not improving development and testing practices)
  - o There may be an improper attitude toward, or expectations of, testing (e.g., not appreciating the value of finding defects during testing)
- **Technical issues:**
  - o Requirements may not be defined well enough
  - o The requirements may not be met, given existing constraints
  - o The test environment may not be ready on time
  - o Data conversion, migration planning, and their tool support may be late
  - o Weaknesses in the development process may impact the consistency or quality of project work products such as design, code, configuration, test data, and test cases
  - o Poor defect management and similar problems may result in accumulated defects and other technical debt
- **Supplier issues:**
  - o A third party may fail to deliver a necessary product or service, or go bankrupt
  - o Contractual issues may cause problems to the project

Project risks may affect both development activities and test activities. In some cases, project managers are responsible for handling all project risks, but it is not unusual for test managers to have responsibility for test-related project risks.

55- Describe, by using examples, how product risk analysis may influence thoroughness and scope of testing (K2)

Risk is used to focus the effort required during testing. It is used to decide where and when to start testing and to identify areas that need more attention. Testing is used to reduce the probability of an adverse event occurring, or to reduce the impact of an adverse event. Testing is used as a risk mitigation activity, to provide feedback about identified risks, as well as providing feedback on residual (unresolved) risks.

A risk-based approach to testing provides proactive opportunities to reduce the levels of product risk. It involves product risk analysis, which includes the identification of product risks and the assessment of each risk's likelihood and impact. The resulting product risk information is used to guide test planning, the specification, preparation and execution of test cases, and test monitoring and control. Analyzing product risks early contributes to the success of a project.

In a risk-based approach, the results of product risk analysis are used to:

- Determine the test techniques to be employed
- Determine the particular levels and types of testing to be performed (e.g., security testing, accessibility testing)
- Determine the extent of testing to be carried out
- Prioritize testing in an attempt to find the critical defects as early as possible
- Determine whether any activities in addition to testing could be employed to reduce risk (e.g., providing training to inexperienced designers)

Risk-based testing draws on the collective knowledge and insight of the project stakeholders to carry out product risk analysis. To ensure that the likelihood of a product failure is minimized, risk management activities provide a disciplined approach to:

- Analyze (and re-evaluate on a regular basis) what can go wrong (risks)
- Determine which risks are important to deal with
- Implement actions to mitigate those risks
- Make contingency plans to deal with the risks should they become actual events

In addition, testing may identify new risks, help to determine what risks should be mitigated, and lower uncertainty about risks.

Defect Management

56- Write a defect report, covering defects found during testing (K3)

Typical defect reports have the following objectives:

▶ Provide developers and other parties with information about any adverse event that occurred, to enable them to identify specific effects, to isolate the problem with a minimal reproducing test, and to correct the potential defect(s), as needed or to otherwise resolve the problem

▶ Provide test managers a means of tracking the quality of the work product and the impact on the testing (e.g., if a lot of defects are reported, the testers will have spent a lot of time reporting them instead of running tests, and there will be more confirmation testing needed)

▶ Provide ideas for development and test process improvement

A defect report filed during dynamic testing typically includes:

▶ An identifier

▶ A title and a short summary of the defect being reported

▶ Date of the defect report, issuing organization, and author

▶ Identification of the test item (configuration item being tested) and environment

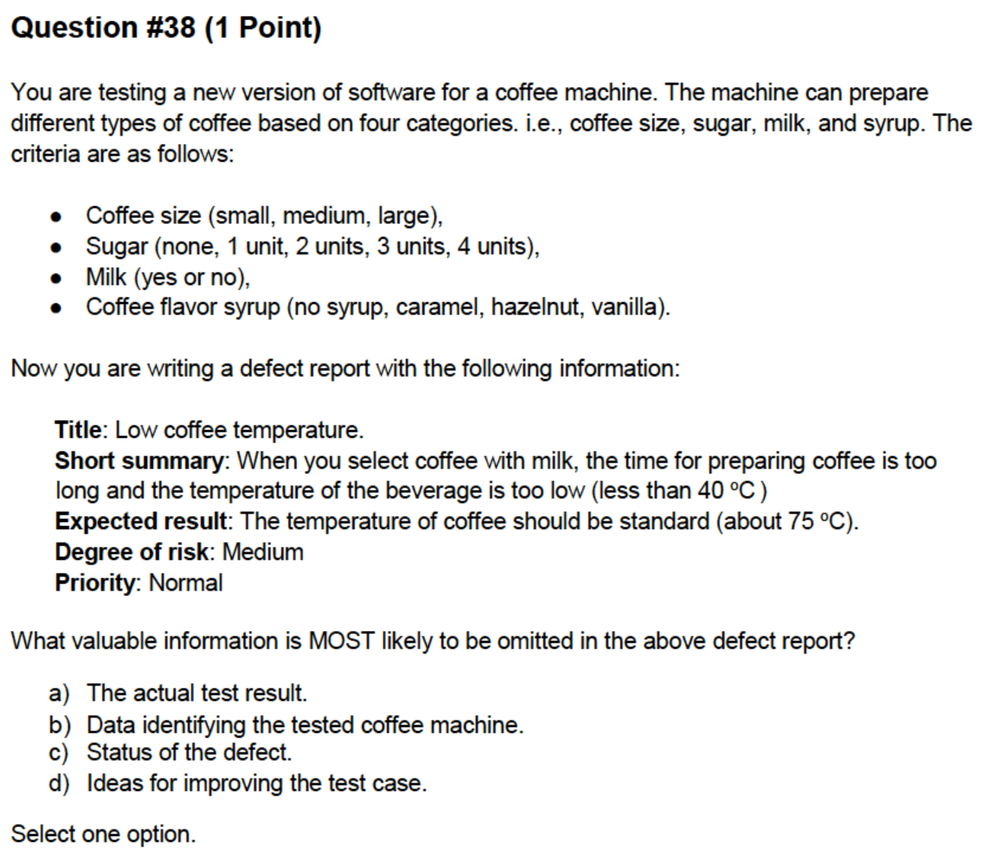
▶ The development lifecycle phase(s) in which the defect was observed

▶ A description of the defect to enable reproduction and resolution, including logs, database dumps screenshots, or recordings (if found during test execution)

▶ Expected and actual results

▶ Scope or degree of impact (severity) of the defect on the interests of stakeholder(s)

▶ Urgency/priority to fix



Question #38 (1 Point)

You are testing a new version of software for a coffee machine. The machine can prepare different types of coffee based on four categories. i.e., coffee size, sugar, milk, and syrup. The criteria are as follows:

- Coffee size (small, medium, large),
- Sugar (none, 1 unit, 2 units, 3 units, 4 units),
- Milk (yes or no),
- Coffee flavor syrup (no syrup, caramel, hazelnut, vanilla).

Now you are writing a defect report with the following information:

Title: Low coffee temperature.
Short summary: When you select coffee with milk, the time for preparing coffee is too long and the temperature of the beverage is too low (less than 40 ºC)
Expected result: The temperature of coffee should be standard (about 75 ºC).
Degree of risk: Medium
Priority: Normal

What valuable information is MOST likely to be omitted in the above defect report?

a) The actual test result.
b) Data identifying the tested coffee machine.
c) Status of the defect.
d) Ideas for improving the test case.

Select one option.

▶ State of the defect report (e.g., open, deferred, duplicate, waiting to be fixed, awaiting confirmation testing, re-opened, closed)

▶ Conclusions, recommendations and approvals

▶ Global issues, such as other areas that may be affected by a change resulting from the defect

▶ Change history, such as the sequence of actions taken by project team members with respect to the defect to isolate, repair, and confirm it as fixed

▶ References, including the test case that revealed the problem

Some of these details may be automatically included and/or managed when using defect management tools, e.g., automatic assignment of an identifier, assignment and update of the defect report state during the workflow, etc. Defects found during static testing, particularly reviews, will normally be documented in a different way, e.g., in review meeting notes.

(Changed to defect management from incident)

Chapter 6 Tool Support for Testing

Test tool considerations

57- Classify test tools according to their purpose and the test activities they support (K2)

Test tools can have one or more of the following purposes depending on the context:

▶ Improve the efficiency of test activities by automating repetitive tasks or tasks that require significant resources when done manually (e.g., test execution, regression testing)

▶ Improve the efficiency of test activities by supporting manual test activities throughout the test process (see section 1.4)

▶ Improve the quality of test activities by allowing for more consistent testing and a higher level of defect reproducibility

▶ Automate activities that cannot be executed manually (e.g., large scale performance testing)

▶ Increase reliability of testing (e.g., by automating large data comparisons or simulating behavior)

Tools can be classified based on several criteria such as purpose, pricing, licensing model (e.g., commercial or open source), and technology used. Tools are classified in this syllabus according to the test activities that they support.

Some tools clearly support only or mainly one activity; others may support more than one activity, but are classified under the activity with which they are most closely associated. Tools from a single provider, especially those that have been designed to work together, may be provided as an integrated suite.

Some types of test tools can be intrusive, which means that they may affect the actual outcome of the test. For example, the actual response times for an application may be different due to the extra instructions that are executed by a performance testing tool, or the amount of code coverage achieved may be distorted due to the use of a coverage tool. The consequence of using intrusive tools is called the probe effect.

Some tools offer support that is typically more appropriate for developers (e.g., tools that are used during component and integration testing). Such tools are marked with "(D)" in the sections below.

Tool support for management of testing and testware

Management tools may apply to any test activities over the entire software development lifecycle.

Examples of tools that support management of testing and testware include:

- Test management tools and application lifecycle management tools (ALM)
- Requirements management tools (e.g., traceability to test objects)
- Defect management tools
- Configuration management tools
- Continuous integration tools (D)

Tool support for static testing

Static testing tools are associated with the activities and benefits described in chapter 3. Examples of such tools include:

- Tools that support reviews
- Static analysis tools (D)

Tool support for test design and implementation

Test design tools aid in the creation of maintainable work products in test design and implementation, including test cases, test procedures and test data. Examples of such tools include:

- Test design tools
- Model-Based testing tools
- Test data preparation tools
- Acceptance test driven development (ATDD) and behavior driven development (BDD) tools
- Test driven development (TDD) tools (D)

In some cases, tools that support test design and implementation may also support test execution and logging, or provide their outputs directly to other tools that support test execution and logging.

Tool support for test execution and logging

Many tools exist to support and enhance test execution and logging activities. Examples of these tools include:

▶ Test execution tools (e.g., to run regression tests)

▶ Coverage tools (e.g., requirements coverage, code coverage (D))

▶ Test harnesses (D)

▶ Unit test framework tools (D)

Tool support for performance measurement and dynamic analysis

Performance measurement and dynamic analysis tools are essential in supporting performance and load testing activities, as these activities cannot effectively be done manually. Examples of these tools include:

▶ Performance testing tools

▶ Monitoring tools

▶ Dynamic analysis tools (D)

Tool support for specialized testing needs

In addition to tools that support the general test process, there are many other tools that support more specific testing issues. Examples of these include tools that focus on:

▶ Data quality assessment

▶ Data conversion and migration

▶ Usability testing

Question #39 (1 Point)

Which one of the following is MOST likely to be a benefit of test execution tools?

a) It is easy to create regression tests.
b) It is easy to maintain version control of test assets.
c) It is easy to design tests for security testing.
d) It is easy to run regression tests.

Select one option.

Question #40 (1 Point)

Which test tool is characterized by the classification below?

1. Tool support for management of testing and testware.
2. Tool support for static testing.
3. Tool support for test execution and logging.
4. Tool support for performance measurement and dynamic analysis.

A. Coverage tools.
B. Configuration management tools.
C. Review tools.
D. Monitoring tools.

a) 1A, 2B, 3D, 4C.
b) 1B, 2C, 3D, 4A.
c) 1A, 2C, 3D, 4B.
d) 1B, 2C, 3A, 4D.

Select one option.

- Accessibility testing
- Localization testing
- Security testing
- Portability testing (e.g., testing software across multiple supported platforms)

(Text simplified and reduced from 2011 to allow easier learnability, understandability)

58- Identify benefits and risks of test automation (K1)

Simply acquiring a tool does not guarantee success. Each new tool introduced into an organization will require effort to achieve real and lasting benefits. There are potential benefits and opportunities with the use of tools in testing, but there are also risks. This is particularly true of test execution tools (which is often referred to as test automation).

Potential benefits of using tools to support test execution include:

- Reduction in repetitive manual work (e.g., running regression tests, environment set up/tear down tasks, re-entering the same test data, and checking against coding standards), thus saving time
- Greater consistency and repeatability (e.g., test data is created in a coherent manner, tests are executed by a tool in the same order with the same frequency, and tests are consistently derived from requirements)
- More objective assessment (e.g., static measures, coverage)
- Easier access to information about testing (e.g., statistics and graphs about test progress, defect rates and performance)

Potential risks of using tools to support testing include:

- Expectations for the tool may be unrealistic (including functionality and ease of use)
- The time, cost and effort for the initial introduction of a tool may be under-estimated (including training and external expertise)
- The time and effort needed to achieve significant and continuing benefits from the tool may be under-estimated (including the need for changes in the test process and continuous improvement in the way the tool is used)
- The effort required to maintain the test assets generated by the tool may be under-estimated
- The tool may be relied on too much (seen as a replacement for test design or execution, or the use of automated testing where manual testing would be better)
- Version control of test assets may be neglected

- ▶ Relationships and interoperability issues between critical tools may be neglected, such as requirements management tools, configuration management tools, defect management tools and tools from multiple vendors
- ▶ The tool vendor may go out of business, retire the tool, or sell the tool to a different vendor
- ▶ The vendor may provide a poor response for support, upgrades, and defect fixes
- ▶ An open source project may be suspended
- ▶ A new platform or technology may not be supported by the tool
- ▶ There may be no clear ownership of the tool (e.g., for mentoring, updates, etc.)

(Reworded and downgraded from K2 to K1)

59- Remember special considerations for test execution and test management tools (K1)

In order to have a smooth and successful implementation, there are a number of things that ought to be considered when selecting and integrating test execution and test management tools into an organization.

Test execution tools

Test execution tools execute test objects using automated test scripts. This type of tool often requires significant effort in order to achieve significant benefits.

Capturing tests by recording the actions of a manual tester seems attractive, but this approach does not scale to large numbers of test scripts. A captured script is a linear representation with specific data and actions as part of each script. This type of script may be unstable when unexpected events occur. The latest generation of these tools, which takes advantage of “smart” image capturing technology, has increased the usefulness of this class of tools, although the generated scripts still require ongoing maintenance as the system’s user interface evolves over time.

A data-driven testing approach separates out the test inputs and expected results, usually into a spreadsheet, and uses a more generic test script that can read the input data and execute the same test script with different data. Testers who are not familiar with the scripting language can then create new test data for these predefined scripts.

In a keyword-driven testing approach, a generic script processes keywords describing the actions to be taken (also called action words), which then calls keyword scripts to process the associated test data.

Testers (even if they are not familiar with the scripting language) can then define tests using the keywords and associated data, which can be tailored to the application being tested. Further details and examples of data-driven and keyword-driven testing approaches are given in ISTQB-TAE Advanced Level Test Automation Engineer Syllabus, Fewster 1999 and Buwalda 2001.

The above approaches require someone to have expertise in the scripting language (testers, developers or specialists in test automation). Regardless of the scripting technique used, the expected results for each test need to be compared to actual results from the test, either dynamically (while the test is running) or stored for later (post-execution) comparison.

Model-Based testing (MBT) tools enable a functional specification to be captured in the form of a model, such as an activity diagram. This task is generally performed by a system designer. The MBT tool interprets the model in order to create test case specifications which can then be saved in a test management tool and/or executed by a test execution tool (see ISTQB-MBT Foundation Level Model- Based Testing Syllabus).

Test management tools

Test management tools often need to interface with other tools or spreadsheets for various reasons, including:

- ▶ To produce useful information in a format that fits the needs of the organization
- ▶ To maintain consistent traceability to requirements in a requirements management tool
- ▶ To link with test object version information in the configuration management tool

This is particularly important to consider when using an integrated tool (e.g., Application Lifecycle Management), which includes a test management module (and possibly a defect management system), as well as other modules (e.g., project schedule and budget information) that are used by different groups within an organization.

(Rephrased "static analysis" is removed from LO compared to 2011 syllabus)

Effective use of Tools

60- Identify the main principles for selecting a tool (K1)

The main considerations in selecting a tool for an organization include:

- ▶ Assessment of the maturity of the organization, its strengths and weaknesses
- ▶ Identification of opportunities for an improved test process supported by tools
- ▶ Understanding of the technologies used by the test object(s), in order to select a tool that is compatible with that technology
- ▶ The build and continuous integration tools already in use within the organization, in order to ensure tool compatibility and integration
- ▶ Evaluation of the tool against clear requirements and objective criteria
- ▶ Consideration of whether or not the tool is available for a free trial period (and for how long)
- ▶ Evaluation of the vendor (including training, support and commercial aspects) or support for noncommercial (e.g., open source) tools

- Identification of internal requirements for coaching and mentoring in the use of the tool
- Evaluation of training needs, considering the testing (and test automation) skills of those who will be working directly with the tool(s)
- Consideration of pros and cons of various licensing models (e.g., commercial or open source)
- Estimation of a cost-benefit ratio based on a concrete business case (if required)

As a final step, a proof-of-concept evaluation should be done to establish whether the tool performs effectively with the software under test and within the current infrastructure or, if necessary, to identify changes needed to that infrastructure to use the tool effectively

(Rewording to match the current content, and make the LO better examinable. Avoiding confusion about introducing tools instead of selecting tools)

61- Recall the objective for using pilot projects to introduce tools (K1)

After completing the tool selection and a successful proof-of-concept, introducing the selected tool into an organization generally starts with a pilot project, which has the following objectives:

- Gaining in-depth knowledge about the tool, understanding both its strengths and weaknesses
- Evaluating how the tool fits with existing processes and practices, and determining what would need to change
- Deciding on standard ways of using, managing, storing, and maintaining the tool and the test assets (e.g., deciding on naming conventions for files and tests, selecting coding standards, creating libraries and defining the modularity of test suites)
- Assessing whether the benefits will be achieved at reasonable cost
- Understanding the metrics that you wish the tool to collect and report, and configuring the tool to ensure these metrics can be captured and reported

(Rewording to match the current content, and make the LO better examinable. Removed proof-of-concept, as the term does not need to be in the syllabus.)

62- Identify the success factors for evaluation, implementation, deployment and on-going support of test tools in an organization (K1)

Success factors for evaluation, implementation, deployment, and on-going support of tools within an organization include:

- Rolling out the tool to the rest of the organization incrementally
- Adapting and improving processes to fit with the use of the tool

- Providing training, coaching, and mentoring for tool users
- Defining guidelines for the use of the tool (e.g., internal standards for automation)
- Implementing a way to gather usage information from the actual use of the tool
- Monitoring tool use and benefits
- Providing support to the users of a given tool
- Gathering lessons learned from all users

It is also important to ensure that the tool is technically and organizationally integrated into the software development lifecycle, which may involve separate organizations responsible for operations and/or third party suppliers.

(LO rephrased to ensure less interpretation and to allow easier learnability.)