DICTIONARY

Ambiguities	неясноти	Vice versa	обратно
Omissions	пропуски	Distinguish	различавам
Inaccuracies	неточности	Iterative	повтарящ се
Contradictions	противоречия	Subsequent	последващ
Superfluous statements	излишни твърдения	Thoroughness	задълбоченост
Testware	тестов софтуер	Susceptibility	чувствителност
Significantly	значително	Exasperation	раздразнение
Bias	пристрастие	Disprove	опровергавам
Bearer	приносител	Adversarial	състезателен
Interpersonal	междуличностни	Rigorous	строг
Tailoring	приспособяване	Curtail	съкратен
Impeding	пречещи	Residual	остатъчен
Unambiguously	недвусмислено	Intrusive	натрапчив

K1 - REMEMBER - 8 questions - suggested 8 min
K2 - UNDERSTAND - 24 questions - suggested 24 min
K3 - APPLY - 8 questions - suggested 24 min

ISO/IEC/IEEE 29119-1 (2013) Software testing - Concepts and definitions
ISO/IEC/IEEE 29119-2 (2013) Software testing - Test processes
ISO/IEC/IEEE 29119-3 (2013) Software testing - Test documentation
ISO/IEC/IEEE 29119-4 (2015) Software testing - Test techniques
ISO/IEC 25010, (2011) Systems and software Quality Requirements and Evaluation (SQuaRE) System and software quality models
ISO/IEC 20246: (2017) Work product reviews

1. FUNDAMENTALS OF TESTING (175 min - 16 pages)
(8 questions - K1 - 2, K2 - 6)

Learning Objectives for Fundamentals of Testing:

1.1 What is Testing?

Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation. Software testing is a process which includes many different activities.
Dynamic testing - involve the execution of the component or system being tested.
Static testing - Reviewing work products such as requirements, user stories, and source code.
Testing involves checking whether the system meets specified requirements, it also involves validation, which is checking whether the system will meet user and other stakeholder needs in its operational environment.

FL-1.1.1 (K1) Identify typical objectives of testing

To prevent defects by evaluating work products such as requirements, user stories, design, and code.
To verify whether all specified requirements have been fulfilled.
To check whether the test object is complete and validate if it works as the users and other stakeholders expect.
To build confidence in the level of quality of the test object.
To find defects and failures by reducing the level of risk of inadequate software quality.
To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object.
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 SDLC model.

FL-1.1.2 (K2) Differentiate testing from debugging

Testing - set of activities conducted to facilitate discovery and/or evaluation of properties of one or more test items.
Debugging - development activity that finds, analyzes, and fixes defects.

1.2 Why is Testing Necessary?

Rigorous testing of components and systems, and their associated documentation, can help reduce the risk of failures occurring during operation.

FL-1.2.1 (K2) Give examples of why testing is necessary

The identification and removal of requirements defects reduces the risk of incorrect or untestable features being developed.
The understanding of design can reduce the risk of fundamental design defects and enable tests to be identified at an early stage.
The understanding of the code can reduce the risk of defects within the code and the tests.
The verification and validation of the software prior to release can detect failures that might otherwise have been missed, and support the process of removing the defects that caused the failures. This increases the likelihood that the software meets stakeholder needs and satisfies requirements.

FL-1.2.2 (K2) Describe the relationship between testing and quality assurance and give examples of how testing contributes to higher quality

Quality management includes all activities that direct and control an organization with regard to quality. 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. 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. Testing contributes to the achievement of quality in a variety of ways.

FL-1.2.3 (K2) Distinguish between error, defect, and failure

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. If a defect in the code is executed, this may cause a failure, but not necessarily in all circumstances. In addition to failures caused due to defects in the code, failures can also be caused by environmental 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.

FL-1.2.4 (K2) Distinguish between the root cause of a defect and its effects

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

The complaint of the defect is the effect.

1.3 Seven Testing Principles

FL-1.3.1 (K2) Explain the seven testing principles

Testing shows presence of defects, not their absence - 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.

Exhaustive testing is impossible - testing everything is not feasible except for trivial cases.

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 SDLC. Testing early in the SDLC helps reduce or eliminate costly changes.

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.

Beware of the pesticide paradox - if the same tests are repeated over and over again, eventually these tests no longer find any new defects.

Testing is context dependent - testing is done differently in different contexts.

Absence of errors is a fallacy - it is a fallacy (a mistaken belief) to expect that just finding and fixing a large number of defects will ensure the success of a system.

1.4 Test Process

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.

FL-1.4.1 (K2) Explain the impact of context on the test process

It is very useful if the test basis has measurable coverage criteria defined. The coverage criteria can act effectively as key performance indicators to drive the activities that demonstrate achievement of software test objectives.

Factors that influence the test process:
- SDLC model and project methodologies being used.
- Test levels and test types being considered.
- Product and project risks.
- Business domain.
- Operational constraints - budgets and resources, timescales, complexity, contractual and regulatory requirements.
- Organizational policies and practices.
- Required internal and external standards.

FL-1.4.2 (K2) Describe the test activities and respective tasks within the test process

Test process consists of the following main groups of activities:

Test planning - involves activities that define the objectives of testing and the approach for meeting test objectives within constraints imposed by the context.

Test monitoring and control - Test monitoring involves the on-going comparison of actual progress against planned progress using any test monitoring metrics defined in the test plan. Test control involves taking actions necessary to meet the objectives of the test plan.

Test analysis - the test basis is analyzed to identify testable features and define associated test conditions. Test analysis determines “what to test”. Test analysis includes the following major activities:
- Analyzing the test basis appropriate to the test level being considered.
- Evaluating the test basis and test items to identify defects of various types.
- 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 level of risks.
- Capturing bi-directional traceability between each element of the test basis and the associated test conditions.

Test design - during the test design, the test conditions are elaborated into high-level test cases, sets of high-level test cases, and other testware. 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, and test cases.

Test implementation - during test implementation, the testware necessary for test execution is created and/or completed including sequencing the test cases into test procedures. 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 automated test scripts.
- Arranging the test suites within a test execution schedule in a way that results in efficient test execution.
- Building the test environment 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.

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 version of the test items or test object, test tools 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.
- Reporting defects based on the failures observed.
- Logging the outcome of test executions.
- Repeating test activities either as a result of action taken for an anomaly, or as part of the planned testing.
- Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test suites.

Test completion - these 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 canceled, an Agile project iteration is finished, 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.

FL-1.4.3 (K2) Differentiate the work products that support the test process

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 as well as exit criteria which will be used during test monitoring and control.

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 analysis work products - include defined and prioritized test conditions, each of which is ideally bi-directionally traceable to the specific elements of the test basis it covers.

Test design work products - include test cases and sets of test cases to exercise the test conditions defined in the test analysis. It is often a good practice to design high-level test cases, without concrete values for input data and expected results.

Test implementation work products - include test procedures and the sequencing of those test procedures, test suites, and a test execution schedule.

Test execution work products - include documentation of the status of individual test cases or test procedures, defect reports, and documentation about which test items, test objects, test tools, and testware were involved in the testing.
Test completion work products - include test summary reports, action items for improvement of subsequent projects or iterations, change requests or product backlog items, and finalized testware.

FL-1.4.4 (K2) Explain the value of maintaining traceability between the test basis and test work products

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. 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.
- 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.

1.5 The Psychology of Testing

Software development, including software testing, involves human beings. Therefore, human psychology has important effects on software testing.

FL-1.5.1 (K1) Identify the psychological factors that influence the success of testing

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.

FL-1.5.2 (K2) Explain the difference between the mindset required for test activities and the mindset required for development activities

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.
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.

2. TESTING THROUGHOUT THE SDLC (100 min - 19 pages)
(5 questions - K1 - 1, K2 - 4)

Learning Objectives for Testing Throughout the Software Development Lifecycle

2.1 Software Development Lifecycle Models

A SDLC 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.

FL-2.1.1 (K2) Explain the relationships between software development activities and test activities in the software development lifecycle

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 as soon as drafts are available.
There are sequential development models and Iterative and incremental development models.
In the Waterfall model, the development activities are completed one after another. In this model, test activities only occur after all other development activities have been completed.
In the V model, the test process is integrated throughout the development process, implementing the principle of early testing. In this model, the execution of tests associated with each test level proceeds sequentially, but in some cases overlapping occurs.
Incremental development involves establishing requirements, designing, building, and testing a system in pieces, which means that the software's features grow incrementally.
Iterative development occurs when groups of features are specified, designed, built, and tested together in a series of cycles, often of a fixed duration.

FL-2.1.2 (K1) Identify reasons why software development lifecycle models must be adapted to the context of project and product characteristics

An appropriate SDLC model should be selected and adapted based on the project goal, the type of product being developed, business priorities, and identified product and project risks. Depending on the context of the project, it may be necessary to combine or reorganize test levels and/or test activities. SDLC models themselves may be combined (different models for backend and frontend).
Reasons:

- Difference in product risks of systems (complex or simple project).
- Many business units can be part of a project or program (combination of sequential and agile development).
- Short time to deliver a product to the market (merge of test levels and/or integration of test types in test levels).

2.2 Test Levels

Test levels are groups of activities that are organized and managed together. For every test level, a suitable test environment is required. The test level are:

- Component Testing
- Integration Testing
- System Testing
- Acceptance Testing

FL-2.2.1 (K2) Compare the different test levels from the perspective of objectives, test basis, test objects, typical defects and failures, and approaches and responsibilities

Component Testing

Component testing focuses on components that are separately testable.

- Objectives
    - Reducing risk

        - Verifying whether the functional and non-functional behaviors of the component are as designed and specified
        - Building confidence in the components quality
        - Finding defects in the component
        - Preventing defects from escaping to higher test levels
    - Test basis
        - Detailed design | Code | Data model | Component specifications
    - Test objects
        - Components, units or modules | Code and data structures
        - Classes | Database modules
    - Typical defects and failures
        - Incorrect functionality | Data flow problems | Incorrect code and logic
    - Specific approaches and responsibilities
        - Components 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.

Integration Testing

Integration testing focuses on interactions between components or systems.

- Objectives
    - 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
    - Preventing defects from escaping to higher test levels
    - Component integration testing focuses on the interactions and interfaces between integrated components. Component integration testing is performed after component testing, and is generally automated.
    - 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. System integration testing may be done after system testing or in parallel with ongoing system test activities.
- Test basis
    - Software and system design | Sequence diagrams
    - Interface and communication protocol specifications
    - Use cases | Architecture at component or system level
    - Workflows | External interface definitions
- Test objects
    - Subsystems | Databases | Infrastructure | APIs | Microservices
- Typical defects and failures
    - for Component integration testing
        - 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
    - for System integration testing
        - Inconsistent message structures between systems
        - Incorrect data, missing data, or incorrect data encoding
        - Interface mismatch
        - Failures in communication between systems
        - Unhandled or improperly handled communication failures between systems
        - Incorrect assumptions about the meaning, units, or boundaries of data being passed between systems
        - Failure to comply with mandatory security regulations
- Specific approaches and responsibilities
    - Component integration tests and system integration tests should concentrate on the integration itself. Component integration testing is often the responsibility of developers. System integration testing is generally the responsibility of testers.

System Testing

System testing focuses on the behavior and capabilities of a whole system or product, often considering end-to-end tasks the system can perform and the non-functional behaviors it exhibits while performing those tasks.

- Objectives
    - 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
    - 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.
- Test basis
    - System and software requirement specifications
    - Risk analysis reports | Use cases | Epics and user stories
    - Models of system behavior | State diagrams | System and user manuals
- Test objects
    - Applications | Hardware/software systems | Operating systems
    - System under test | System configuration and configuration data
- Typical defects and failures
    - 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 system environments
        - Failure of the system to work as described in system and user manuals
    - Specific approaches and responsibilities
        - 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 for the aspects of the system to be tested. System testing is typically carried out by independent testers who rely heavily on specifications.

Acceptance Testing

Acceptance testing. like system testing, typically focuses on the behavior and capabilities of a whole system or product.

- Objectives
    - 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
    - User acceptance testing (UAT) - typically focused on validating the fitness for use of the system by intended users in a real or simulated operational environment.
    - Operational acceptance testing (OAT) - it's usually performed in a simulated production environment. Test focus 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;
    - Contractual and regulatory acceptance testing - it's performed against a contract's acceptance criteria for producing custom-developed software. 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 results being witnessed or audited by regulatory agencies.
    - Alpha and beta testing - 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 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..
- Test basis
    - Business processes | User or business requirements
    - Regulations, legal contracts and standards
    - Use cases and/or user stories | System requirements
    - System or user documentation | Installation procedures | Risk analysis reports
    - Test basis for OAT - backup and restore procedures; disaster recovery procedures; non-functional requirements; operations documentation; deployment and installation instructions; performance targets; database packages; security standards or regulations;

- Test objects
    - System under test | System configuration and configuration data
    - Business processes for a fully integrated system | Forms
    - Recovery systems and hot sites | Existing and converted production data
    - Operational and maintenance processes | Reports
- Typical defects and failures
    - 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. In iterative development, project teams can employ various forms of acceptance testing during and at the end of each iteration. All different acceptance testings may occur, either at the close of each iteration, after the completion of each iteration, or after a series of iterations.

2.3 Test Types

A test type is a group of test activities aimed at testing specific characteristics of a software system, or a part of a system, based on specific test objectives.

FL-2.3.1 (K2) Compare functional, non-functional, and white-box testing

Functional Testing - involves tests that evaluate functions that the system should perform. The functions are “what” the system should do. Functional tests should be performed at all test levels, though the focus is different at each level. 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. The thoroughness of functional testing can be measured through functional coverage. Functional coverage is the extent to which some functionality has been exercised by tests, and is expressed as a percentage of the types of element being covered. Functional test design and execution may involve special skills or knowledge, such as knowledge of the particular business problem the software solves.

Non-Functional Testing - evaluates characteristics of systems and software such as usability, performance efficiency or security. Non-functional testing is the testing of “how well” the system behaves. 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 may be used to derive test conditions and test cases for non-functional testing. The thoroughness of non-functional testing can be measured through non-functional coverage. Non-functional 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 types of element being covered. Non-functional test design and execution may involve special skills or knowledge, such as knowledge of the inherent weaknesses of a design or technology or the particular user base.

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

FL-2.3.2 (K1) Recognize that functional, non-functional, and white-box tests occur at any test level

FL-2.3.3 (K2) Compare the purposes of confirmation testing and regression testing

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 to cover changes needed to fix the defect. At the very least, the steps to reproduce the failures 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 intended side-effects are called regressions. Regression testing involves running tests to detect such unintended side-effects.

2.4 Maintenance Testing

Once deployed to production environments, software and systems need to be maintained. When any changes are made as part of maintenance, maintenance testing should be performed, both to evaluate the success with which the changes were made and to check for possible side-effects in parts of the system that remain unchanged. Maintenance can involve planned releases and unplanned releases (hot fixes). A maintenance release may require maintenance testing at multiple test levels, using various test types, based on its scope.

FL-2.4.1 (K2) Summarize triggers for maintenance testing

Modification, such as planned enhancements, corrective and emergence changes, changes of the operational environment, 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.
- Testing restore/retrieve procedures after archiving for long retention periods may also be needed.
- Regression testing may be needed to ensure that any functionality that remains in service still works.

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.

FL-2.4.2 (K2) Describe the role of impact analysis in maintenance testing

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. 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.

3. STATIC TESTING (135 min - 9 pages) (5 questions - K1 - 1, K2 - 3, K3 - 1)

Learning Objectives for Static Testing

3.1 Static Testing Basics

Static testing relies on the manual examination of work products (i.e., reviews) or tool-driven evaluation of the code or other work products (i.e., static analysis). Both types of static testing assess the code or other work product being tested without actually executing the code or work product being tested.

FL-3.1.1 (K1) Recognize types of software work products that can be examined by the different static testing techniques

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 | User guides | Web pages | Configuration set up and infrastructure set up
- Testware, including test plans, test cases, test procedures, and automated test scripts
- Contracts, project plans, schedules, and budget planning
- Models, such as activity diagrams, which may be used for Model-Based testing

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.

FL-3.1.2 (K2) Use examples to describe the value of static testing

- 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
- Reducing development and 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

FL-3.1.3 (K2) 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 lifecycle

Static testing and dynamic testing can have the same objectives 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. 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.

3.2 Review Process

Informal reviews are characterized by not following a defined process and not having formal documented output.

Formal reviews are characterized by team participation, documented results of the review, and documented procedures for conducting the review.

FL-3.2.1 (K2) Summarize the activities of the work product review process

Planning

- defining the scope | estimating effort and time frame
- identifying review characteristics | selecting the people to participate
- defining the entry and exit criteria | checking that entry criteria are met

Initiate review

- distributing the work product and other material
- explaining the scope, objectives, process, roles, and work products to the participants
- answering any questions that participants may have

Individual review

- reviewing all or part of the work product
- noting potential defects, recommendations, and questions

Issue communication and analysis

- communicating identified potential defects
- analyzing potential defects, assigning ownership and status
- evaluating and documenting quality characteristics
- evaluating the review findings against the exit criteria to make a review decision

Fixing and reporting

- creating defect reports for those findings that require changes to a work product
- fixing defects found in the work product reviewed | gathering metrics
- communicating defects to the appropriate person or team
- recording updated status of defects | checking that exit criteria are met
- accepting the work product when the exit criteria are reached

FL-3.2.2 (K1) Recognize the different roles and responsibilities in a formal review

Author

- creates the work product under review
- fixes defects in the work product under review

Management

- 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 (moderator)

- ensures effective running of review meetings
- 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 subjects matter experts, persons working on the project, stakeholders with an interest in the work products, and/or individuals with specific technical or business backgrounds
- identify potential defects in the work product under review
- may represent different perspectives

Scribe (recorder)

- collates potential defects found during the individual review activity
- records new potential defects, open points, and decisions from the review meeting

FL-3.2.3 (K2) Explain the differences between different review types: informal review, walkthrough, technical review, and inspection

Informal review

- Main purpose: detecting potential defects
- Possible additional purposes: generating new ideas or solutions, quickly solving minor problems
- not based on a formal process
- may not involve a review meeting
- may be performed by a colleague of the author
- results may be documented
- varies in usefulness depending on the reviewers
- use of checklists is optional
- very commonly used in Agile

Walkthrough

- Main purpose: find defects, improve the software product, consider alternative implementations, evaluate conformance to standards and specifications
- Possible additional purposes: exchanging ideas about techniques or style variations, training of participants, achieving consensus
- individual preparation before the review meeting is optional
- review meeting is typically led by the author
- scribe is mandatory
- use of checklists is optional
- may take the form of scenarios, dry runs, or simulations
- potential defect logs and review reports are produced
- may vary in practice from quite informal to very formal

Technical review

- Main purpose: gaining consensus, detecting potential defects
- Possible further purposes: evaluating quality and building confidence in the work product, generating new ideas, motivating and enabling authors to improve future work products
- reviewers should be technical peers of the author
- individual preparation before the review meeting is required
- review meeting is optional, ideally led by e trained facilitator
- scribe is mandatory
- use of checklists is optional
- potential defect logs and review reports are produced

Inspection

- Main purpose: detecting potential defects, evaluating quality and building confidence in the work product, preventing future similar defects through author learning and root cause analysis
- Possible further purposes: motivating and enabling authors to improve future work products and the software development process, achieving consensus
- follows a defined process with formal documented outputs, based on rules and checklists
- uses clearly defined roles, and may include a dedicated reader
- individual preparation before the review meeting is required
- reviewers are either peers of the author or experts in other disciplines that are relevant to the work product
- specified entry and exit criteria are used
- scribe is mandatory
- review meeting is led by a trained facilitator
- author cannot act as the review leader, reader, or scribe
- potential defect logs and review report are produced
- metrics are collected and used to improve the entire software development process, including the inspection process

FL-3.2.4 (K3) Apply a review technique to a work product to find defects

Ad hoc - 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.

Checklist-based - a systematic technique, whereby the reviewers detect issues based on checklists that are distributed at a review initiation. A review checklist consists of a set of questions based on potential defects, which may be derived from experience. Checklist 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.

Scenarios and dry runs - reviewers are provided with structured guidelines on how to read through the work product. A scenario-based review supports reviewers in performing “dry runs” on the work product based on expected usage of the work product.

Perspective-based - reviewers take on different stakeholder viewpoints in individual reviewing. Typical stakeholder viewpoints include end user, marketing, designer, tester, or operations. 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. Studies have shown perspective-based reading to be the most effective general technique for reviewing requirements and technical work products.

Role-based - the reviewers evaluate the work product from the perspective of individual stakeholder roles. Typical roles include specific end user types and specific roles in the organization.

FL-3.2.5 (K2) Explain the factors that contribute to a successful review

Organizational success factors:

- each review has clear objectives, defined during review planning, and used as a 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, 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
- participants have adequate time to prepare
- reviews are scheduled with adequate notice
- management supports the review process
- reviews are integrated in the company's quality and/or test policies

People-related success factors:

- the right people are involved to meet the review objectives
- testers are seen as valued reviewers who contribute to the review and learn about the work product
- participants dedicate adequate time and attention to detail
- reviews are conducted on small chunks
- defects found are acknowledged, appreciated, and handled objectively
- the meeting is well-managed, so that participants consider it valuable use of their time
- the review is conducted in an atmosphere of trust
- participants avoid body language and behaviors that might indicate boredom, exasperation, or hostility
- adequate training is provided
- a culture of learning and process improvement is promoted

4. TEST TECHNIQUES (330 min - 7 pages) (11 questions - K1 - 1, K2 - 5, K3 - 5)

Learning Objectives for Test Techniques

4.1 Categories of Test Techniques

The purpose of a test technique, including those discussed in this section, is to help in identifying test conditions, test cases, and test data.

FL-4.1.1 (K2) Explain the characteristics, commonalities, and differences between black-box test techniques, white-box test techniques, and experience-based test techniques

Black-box test technique - based on an analysis of the appropriate test basis.

- Test conditions, test cases, and test data are derived from 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 test basis

White-box test technique - based on an analysis of the architecture, detailed design, internal structure, or the code of the test object.

- Test conditions, test cases, and test data are derived from 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 and the technique applied to the test basis

Experience-based test technique - leverage the experience of developers, testers and users to design, implement, and execute tests.

- Test conditions, test cases, and test data are derived from knowledge and experience of testers, developers, users and other stakeholders

4.2 Black-box Test Techniques

FL-4.2.1 (K3) Apply equivalence partitioning to derive test cases from given requirements

Equivalence partitioning divides data into partitions in such a way that all the members of a given partition are expected to be processed in the same way.

- Valid values are values that should be accepted by the component or system
- Invalid values are values that should be rejected by the component or system

To achieve 100% coverage, test cases must cover all identified partitions by using a minimum of one value from each partition. Coverage is measured as the number of equivalence partitions tested by at least one value, divided by the total number of identified equivalence partitions.

FL-4.2.2 (K3) Apply boundary value analysis to derive test cases from given requirements

Boundary value analysis is an extension of equivalence partitioning, but can only be used when the partition is ordered, consisting of numeric or sequential data. The minimum and maximum values of a partition are its boundary values.

Boundary coverage for a partition is measured as the number of boundary values tested, divided by the total number of identified boundary test values.

FL-4.2.3 (K3) Apply decision table testing to derive test cases from given requirements

Decision tables are a good way to record complex business rules that a system must implement. When creating decision tables, the tester identifies conditions (inputs) and the resulting actions (outputs) of the system.

Common notation for conditions:

- Y (T or 1) means the condition is true
- N (F or 0) means the condition is false
- - (N/A) means the value of the condition doesn't matter

Common notation for actions:

- X (Y or T or 1) means the action should occur
- Blank (N or F or 0) means the action should not occur

The common minimum coverage standard for decision table testing is to have at least one test case per decision rule in the table. This typically involves covering all combinations of conditions. Coverage is measured as the number of decision rules tested by at least one test case, divided by the total number of decision rules.

FL-4.2.4 (K3) Apply state transition testing to derive test cases from given requirements

A state transition diagram shows the possible software states, as well as how the software enters, exits, and transitions between states. A transition is initiated by an event (e.g., user input of a value into a field). The same event can result in two or more different transitions from the same state. The state change may result in the software taking an action. Tests can be designed to cover a typical sequence of states, to exercise all states, to exercise every transition, to exercise specific sequences of transitions, or to test invalid transitions.

Coverage is commonly measured as the number of identified states or transitions tested, divided by the total number of identified states or transitions in the test object.

FL-4.2.5 (K2) Explain how to derive test cases from a use case

Tests can be derived from use cases, which are a specific way of designing interactions with software items. They incorporate requirements for the software functions. 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. Coverage can be measured by the number of use case behaviors tested divided by the total number of use case behaviors.

4.3 White-box Test Techniques

White-box testing is based on the internal structure of the test object.

FL-4.3.1 (K2) Explain statement coverage

Statement testing exercises the potential 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.

FL-4.3.2 (K2) Explain decision coverage

Decision testing exercises the decisions in the code and tests the code that is executed based on the decision outcomes. For an IF statement, there is 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 one.
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.

FL-4.3.3 (K2) Explain the value of statement and decision coverage

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.
When 100% decision coverage is achieved, it executes all decision outcomes, which includes testing true outcome and also the false outcome, even when there is no explicit false statement.
100% decision coverage guarantees 100% statement coverage.

4.4 Experience-based Test Techniques

When applying experience-based test techniques, the test cases are derived from the tester’s skill and intuition, and their experience with similar applications and technologies.

FL-4.4.1 (K2) Explain error guessing

Error guessing is a technique used to anticipate the occurrence of errors, defects, and failures, based on the tester’s knowledge, including how the application has worked in the past, what kind of errors tend to be made, failures that have occurred in other applications

FL-4.4.2 (K2) Explain exploratory testing

In exploratory testing, informal tests are designed, executed, logged, and evaluated dynamically during test execution. Exploratory testing is most useful when there are few or inadequate specifications or significant time pressure on testing.

FL-4.4.3 (K2) Explain checklist-based testing

In checklist-based testing, testers design, implement, and execute tests to cover test conditions found in a checklist. 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.

5. TEST MANAGEMENT (225 min - 15 pages)
(9 questions - K1 - 2, K2 - 5, K3 - 2)

Learning Objectives for Test Management

5.1 Test Organization

FL-5.1.1 (K2) Explain the benefits and drawbacks of independent testing

Benefits of test independence:

- 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
- Independent testers of a vendor can report in an upright and objective manner about the system under test without (political) pressure of the company that hired them

Drawbacks of test independence:

- Isolation from the development team, may lead 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
- Independent testers may lack some important information

FL-5.1.2 (K1) Identify the tasks of a test manager and tester

Test Manager tasks:

- 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
- coordinate the test plan 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 (definition of done) and facilitate test completion activities
- prepare and deliver test progress reports and test summary reports based on the information gathered
- adapt planning based on test results and progress 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, allocating time and effort for pilot projects, and providing continuing support in the use of the tools
- decide about the implementation of test environment

- promote and advocate the testers, the test team, and the test profession within the organization
- develop the skills and careers of testers

Tester tasks:

- review and contribute to test plans
- analyze, review, and assess requirements, user stories and acceptance criteria, specifications, and models for testability
- identify and document test conditions, and capture traceability between test cases, test conditions, and the test basis
- design, set up, and verify test environment, 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
- evaluate non-functional characteristics such as performance efficiency, reliability, usability, security, compatibility, and portability
- review tests developed by others

5.2 Test Planning and Estimation

FL-5.2.1 (K2) Summarize the purpose and content of a test plan

A test plan outlines test activities for development and maintenance projects. Test planning is a continuous activity and is performed throughout the product's lifecycle. Test planning activities may include the following and some of these may be documented in a test plan:

- determining 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 or in the context of each iteration
- selecting metrics for test monitoring and control
- budgeting for the test activities
- determining the level of detail and structure for test documentation

The content of test plans vary, and can extend beyond the topics identified above.

FL-5.2.2 (K2) Differentiate between various test strategies

Analytical: based on an analysis of some factor (e.g. requirement or risk).

Model-Based: 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).

Methodical: 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 (standard-compliant): 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 (consultative): 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: motivated by a desire to avoid regression of existing capabilities. Includes reuse of existing testware, extensive automation of regression tests, and standard test suites.
Reactive: testing is reactive to the component or system being tested, and the events occurring during test execution, rather than being pre-planned. Tests are designed and implemented, and may immediately be executed in response to knowledge gained from prior test results.

An appropriate test strategy is often created by combining several of these types of test strategies.

FL-5.2.3 (K2) Give examples of potential entry and exit criteria

Entry criteria:

- availability of testable requirements, user stories, and/or models
- 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

Exit criteria:

- planned tests have been executed
- a defined level of coverage 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

FL-5.2.4 (K3) Apply knowledge of prioritization, and technical and logical dependencies, to schedule test execution for a given set of test cases

Once the various test cases and test procedures are produced 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.

FL-5.2.5 (K1) Identify factors that influence the effort related to testing

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
- 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
- team cohesion and leadership

Test results:

- the number and severity of defects found | the amount of rework required

FL-5.2.6 (K2) Explain the difference between two estimation techniques: the metrics-based technique and the expert-based technique

Metrics-based technique: estimating the test effort based on metrics of former similar projects, or based on typical values

Expert-based technique: estimating the test effort based on the experience of the owners of the testing tasks or by experts

5.3 Test Monitoring and Control

The purpose of test monitoring is to gather information and provide feedback and visibility about test activities.

Test control describes any guiding or corrective actions taken as a result of information and metrics gathered and reported.

FL-5.3.1 (K1) Recall metrics used for testing

Common test metrics:

- percentage of planned work done in test case preparation
- 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)
- defect information (e.g. defect density, defects found and fixed)
- 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 or running the next test

FL-5.3.2 (K2) Summarize the purposes, contents, and audiences for test reports

The purpose of test reporting is to summarize and communicate test activity information, both during and at the end of a test activity. 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.

Typical test summary report 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
- residual risks | reusable test work products produced

The contents of a test report will vary depending on the project, the organizational requirements, and the SDLC.

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.

5.4 Configuration Management

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.

FL-5.4.1 (K2) Summarize how configuration management supports testing

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 items 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 should be identified and implemented.

5.5 Risks and Testing

FL-5.5.1 (K1) Define risk level by using likelihood and impact

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 from that event.

FL-5.5.2 (K2) Distinguish between project and product risks

Product risk involves the possibility that a work product may fail to satisfy the legitimate needs of its users and/or stakeholders.

Product risks:

- 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 requirements
- 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 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.

Project risks:

- Project issues:
    - delays may occur in delivery, task completion, or satisfaction of exit criteria or definition of done
    - inaccurate estimates, reallocation of funds to higher priority projects, or general cost-cutting across the organization may result in inadequate funding
    - late changes may result in substantial re-work
- Organizational issues:
    - skills, training, and staff may not be sufficient
    - personnel issues may cause conflict and problems
    - users, business staff, or subject matter experts may not be available due to conflicting business priorities

- Political issues:
    - testers may not communicate their needs and/or the test results adequately
    - developers and/or testers may fail to follow up on information found in testing and reviews
    - there may be an improper attitude toward, or expectations of, testing appreciating the value of finding defects during testing
- Technical issues:
    - requirements may not be defined well enough
    - the requirements may not be met, given existing constraints
    - the test environment may not be ready on time
    - data conversion, migration planning, and their tool support may be late
    - 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
    - poor defect management and similar problems may result in accumulated defects and other technical debt
- Supplier issues:
    - a third party may fail to deliver a necessary product or service, or go bankrupt
    - contractual issues may cause problems to the project

FL-5.5.3 (K2) Describe, by using examples, how product risk analysis may influence the thoroughness and scope of testing

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
- 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

5.6 Defect Management

Since one of the objectives of testing is to find defects, defects found during testing should be logged. The way in which defects are logged may vary, depending on the context of the component or system being tested, the test level, and the SDLC model. Any defects identified should be investigated and should be tracked from discovery and classification to their resolution. In order to manage all defects to resolution, an organization should establish a defect management process which includes a workflow and rules for classification. This process must be agreed with all those participating in defect management, including architects, designers, developers, testers, and product owners.

FL-5.6.1 (K3) Write a defect report, covering a defect found during testing

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 minimal reproducing test, and to correct the potential defect, 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

Defect report includes:

- an identifier | a title and a short summary
- date of the defect report, issuing organization, and author

- identification of the test item and environment
- the development lifecycle phase in which the defect was observed
- a description of the defect to enable reproduction and resolution, including logs, database dumps, screenshots, or recordings
- expected and actual results
- scope or degree of impact of the defect on the interests of stakeholders
- urgency/priority to fix | state of the defect report
- 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

6. TOOL SUPPORT FOR TESTING (40 min - 6 pages) (2 questions - K1 - 1, K2 - 1)

Learning Objectives for Test Tools

6.1 Test tool considerations

Test tools can be used to support one or more testing activities. Such tools include:

- tools that are directly used in testing, such as test execution tools and test data preparation tools
- tools that help to manage requirements, test cases, test procedures, automated test scripts, test results, test data, and defects, and for reporting and monitoring test execution
- tools that are used for analysis and evaluation | any tool that assists in testing

FL-6.1.1 (K2) Classify test tools according to their purpose and the test activities they support

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
- improve the efficiency of test activities by supporting manual test activities throughout the test process
- 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
- increase reliability of testing

Some tools offer support that is typically more appropriate for developers. Such tools are marked with “D” in the sections below.

Tool support for management of testing and testware

- test management tools and application lifecycle management tools
- requirements management tools | defect management tools
- configuration management tools | continuous integration tools (D)

Tool support for static testing

- static analysis tools (D)

Tool support for test design and implementation

- Model-Based testing tools | test data preparation tools

Tool support for test execution and logging

- test execution tools | coverage tools (D) | test harnesses (D)

Tool support for performance measurement and dynamic analysis

- performance testing tools | dynamic analysis tools (D)

Tool support for specialized testing needs

FL-6.1.2 (K1) Identify benefits and risks of test automation

Benefits:

- reduction in repetitive manual work
- greater consistency and repeatability
- more objective assessment
- easier access to information about testing

Risks:

- expectations for the tool may be unrealistic
- the time, cost and effort for the initial introduction of a tool may be under-estimated
- the time and effort needed to achieve significant and continuing benefits from the tool may be under-estimated
- the effort required to maintain the test work products generated by the tool may be under-estimated
- the tool may be relied on too much
- version control of test work products 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

FL-6.1.3 (K1) Remember special considerations for test execution and test management tools

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

- Capturing test approach - 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, and require ongoing maintenance as the system's user interface evolves over time.
- Data-driven test approach - this test 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 test script with different data.
- Keyword-driven test approach - this test 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.

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

6.2 Effective use of tools

FL-6.2.1 (K1) Identify the main principles for selecting a tool

Main principles for tool selection:

- assessment of the maturity of the own organization, its strengths and weaknesses
- identification of opportunities for an improved test process supported by tools
- understanding of the technologies used by the test objects
- understanding 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
- evaluation of the vendor or support for non-commercial tools
- identification of internal requirements for coaching and mentoring in the use of the tool
- evaluation of training needs, considering the testing skills of those who will be working directly with the tools
- consideration of pros and cons of various licensing models
- estimation of a cost-benefit ratio based on a concrete business case

As a final step, a proof-of-concept (POC) 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.

FL-6.2.2 (K1) Recall the objectives for using pilot projects to introduce tools

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 work products
- 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

FL-6.2.3 (K1) Identify the success factors for evaluation, implementation, deployment, and on-going support of test tools in an organization

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
- 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