International Software Testing Qualifications Board (ISTQB)

Certified Tester Foundation Level (CTFL) Syllabus Version 2018 v3.1.1

Dian Permana and QA Team

Learning together at AVANA

April 3, 2023

Overview

1. Fundamental of testing
2. Testing Throughout the Software Development Lifecycle
3. Static Testing
4. Test Techniques
5. Test Management
6. Tool Support for Testing

Fundamental of testing

1.1 What is Testing

problem

Most people have had an experience with software that did not work as expected. Software that does not work correctly can lead to many problems, including loss of money, time, or business reputation, and even injury or death

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

A common misperception of testing

- it only consists of running tests, i.e., executing the software and checking the results
- it focuses entirely on verification of requirements, user stories, or other specifications. While testing does involve 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(s).

1.1 What is Testing

dynamic testing

Some testing does involve the execution of the component or system being tested;

static testing

Other testing does not involve the execution of the component or system being tested

1.1 What is Testing

Typical Objectives of Testing

- To prevent defects by evaluate 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 comply with contractual, legal, or regulatory requirements or standards, and/or to verify the test object's compliance with such requirements or standards

1.1 What is Testing

Testing

Executing tests can show failures that are caused by defects in the software.

Debugging

The development activity that finds, analyzes, and fixes such defects. Subsequent confirmation testing checks whether the fixes resolved the defects.

1.2 Why is Testing Necessary?

1.2.1 Testing's Contributions to Success
1.2.2 Quality Assurance and Testing
1.2.3 Errors, Defects, and Failures
1.2.4 Defects, Root Causes and Effects

1.2.1 Testing's Contributions to Success

- Having testers involved in requirements reviews or user story refinement could detect defects in these work products. The identification and removal of requirements defects reduces the risk of incorrect or untestable features being developed.
- Having testers work closely with system designers while the system is being designed can increase each party's understanding of the design and how to test it. This increased understanding can reduce the risk of fundamental design defects and enable tests to be identified at an early stage.

1.2.1 Testing's Contributions to Success

- Having testers work closely with developers while the code is under development can increase each party's understanding of the code and how to test it. This increased understanding can reduce the risk of defects within the code and the tests.
- Having testers verify and validate 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 (i.e., debugging). This increases the likelihood that the software meets stakeholder needs and satisfies requirements

1.2.2 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

1.2.2 Quality Assurance and Testing

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

As described in sections 1.1.1 and 1.2.1, Testing contributes to the achievement of quality in a variety of ways

1.2.3 Errors, Defects, and Failures

Errors

A person can make an **error** (mistake)

Defects

which can lead to the introduction of a **defect** (fault or bug) in the software code or in some other related work product

Failures

If a **defect** in the code is executed, this may cause a **failure**, but not necessarily in all circumstances

1.2.4 Defects, Root Causes and Effects

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

1.2.4 Defects, Root Causes and Effects

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 an error while writing the user story

1.3 Seven Testing Principles

- Testing shows the presence of defects, not their absence
- Exhaustive testing is impossible
- Early testing saves time and money
- Defects cluster together
- Beware of the pesticide paradox
- Testing is context dependent
- Absence-of-errors is a fallacy

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

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

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

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

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

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 software development lifecycle project

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

1.4 Test Process

1.4.1 Test Process in Context
1.4.2 Test Activities and Tasks
1.4.3 Test Work Products
1.4.4 Traceability between the Test Basis and Test Work Products

1.4.1 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:
  - Budgets and resources
  - Timescales
  - Complexity
  - Contractual and regulatory requirements
- Organizational policies and practices
- Required internal and external standards

1.4.2 Test Activities and Tasks

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

1.4.3 Test Work Products

- Test planning work products
- Test monitoring and control work products
- Test analysis work products
- Test design work products
- Test implementation work products
- Test execution work products

1.4.4 Traceability between the Test Basis and Test Work Products

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

1.5 The Psychology of Testing

1.5.1 Human Psychology and Testing

1.5.2 Tester's and Developer's Mindsets

1.5.1 Human Psychology and Testing

- 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.

1.5.2 Tester's and Developer's Mindsets

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

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 become aware of errors committed by themselves.

Testing Throughout the Software Development Lifecycle

2.1 Software Development Lifecycle Models
2.2 Test Levels
2.3 Test Types
2.4 Maintenance Testing

2.1 Software Development and Software Testing

Sequential development models 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.

Iterative and incremental development models

- 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

2.2 Test Levels

	Objectives of component testing	Test basis	Test objects	Typical defects and failures
Component testing	• 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	Detailed design • Code • Data model • Component specifications	• Components, units or modules • Code and data structures • Classes • Database modules	• Incorrect functionality (e.g., not as described in design specifications) • Data flow problems • Incorrect code and logic
Integration testing	• 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	• Software and system design • Sequence diagrams • Interface and communication protocol specifications • Use cases • Architecture at component or system level • Workflows • External interface definitions	• Subsystems • Databases • Infrastructure • Interfaces • APIs • Microservices	• 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
System testing	• 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	• 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	• Applications • Hardware/software systems • Operating systems • System under test (SUT) • System configuration and configuration data	• 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 environment(s) • Failure of the system to work as described in system and user manuals
Acceptance testing	• 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	• 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	• 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	• 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

2.3 Test Type

Functional Testing	1). Functional testing of a system involves tests that evaluate functions that the system should perform. 2). 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 3). Functional tests should be performed at all test levels (e.g tests for components may be based on a component specification)
Non-functional Testing	1). Non-functional testing is the testing of "how well" the system behaves.
White-box Testing	1). White-box testing derives tests based on the system's internal structure or implementation 2). Internal structure may include code, architecture, work flows, and/or data flows within the system 3). White-box test design and execution may involve special skills or knowledge, such as the way the code is built, how data is stored (e.g., to evaluate possible database queries), and how to use coverage tools and to correctly interpret their results
Change-related Testing	1). 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 2). 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 3). Confirmation testing and regression testing are performed at all test levels
Test Types and Test Levels	All of above

2.4 Maintenance Testing

2.4.1 Triggers for Maintenance

2.4.2 Impact Analysis for Maintenance

2.4.1 Triggers for Maintenance

- 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

2.4.2 Impact Analysis for Maintenance

- 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
- The people involved do not have domain and/or system knowledge

Static Testing

3.1 Static Testing Basics

3.1.1 Work Products that Can Be Examined by Static Testing
3.1.2 Benefits of Static Testing
3.1.3 Differences between Static and Dynamic Testing

3.1.1 Work Products that Can Be Examined by Static Testing

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

3.1.1 Work Products that Can Be Examined by Static Testing

- Web pages
- Contracts, project plans, schedules, and budget planning
- Configuration set up and infrastructure set up
- Models, such as activity diagrams, which may be used for Model-Based testing (see ISTQBCTFL-MBT and Kramer 2016)

3.1.2 Benefits 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 (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

3.1.3 Differences between Static and Dynamic Testing

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)

3.2 Review Process

3.2.1 Work Product Review Process
3.2.2 Roles and responsibilities in a formal review
3.2.3 Review Types
3.2.4 Applying Review Techniques
3.2.5 Success Factors for Reviews

3.2.1 Work Product Review Process

The review process comprises the following main activities:

- Planning
- Initiate review
- Individual review (i.e., individual preparation)
- Issue communication and analysis
- Fixing and reporting

3.2.2 Roles and responsibilities in a formal review

A typical formal review will include the roles below:

- Author
- Management
- Facilitator (often called moderator)
- Review leader
- Reviewers
- Scribe (or recorder)

3.2.3 Review Types

A typical formal review will include the roles below:

- Informal review (e.g., buddy check, pairing, pair review)
- Walkthrough
- Technical review
- Inspection

3.2.4 Applying Review Techniques

- Ad hoc
- Checklist-based
- Scenarios and dry runs
- Perspective-based
- Role-based

3.2.5 Success Factors for Reviews

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
- Reviews are integrated in the company's quality and/or test policies.

3.2.5 Success Factors for Reviews

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
- etc

Test Techniques

4.1 Categories of Test Techniques.
4.2 Black-box Test Techniques.
4.3 White-box Test Techniques.
4.4 Experience-based Test Techniques

4.1 Categories of Test Techniques

- Component or system complexity
- Regulatory standards
- Customer or contractual requirements
- Risk levels and types
- Available documentation
- Tester knowledge and skills
- Available tools
- Time and budget
- Software development lifecycle model
- The types of defects expected in the component or system

4.2 Black-box, 4.3 white-box, 4.4 Experience-based

Black-box

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)

White-box

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

Experience-based

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

Black-box Test

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing
- Use Case Testing

White-box Test

- Statement Testing and Coverage
- Decision Testing and Coverage
- The Value of Statement and Decision Testing

Experience-based Test

- Error Guessing
- Exploratory Testing
- Checklist-based Testing

Test Management

5.1 Test Organization.
5.2 Test Planning and Estimation.
5.3 Test Monitoring and Control.
5.4 Configuration Management
5.5 Risks and Testing
5.6 Defect Management

5.1 Test Organization

5.1.1 Independent Testing

5.1.2 Tasks of a Test Manager and Tester

5.1.1 Independent Testing

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
- 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

5.1.2 Tasks of a Test Manager and Tester

Test Manager

- 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

Tester

- 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)

5.1.2 Tasks of a Test Manager and Tester

Test Manager

- 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

Tester

- 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

5.1.2 Tasks of a Test Manager and Tester

Test Manager

- 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) and facilitate test completion activities
- Prepare and deliver test progress reports and test summary reports based on the information gathered

Tester

- 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)

5.1.2 Tasks of a Test Manager and Tester

Test Manager

- 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.)

Tester

- 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

5.2.1 Purpose and Content of a Test Plan
5.2.2 Test Strategy and Test Approach
5.2.3 Entry Criteria and Exit Criteria (Definition of Ready and Definition of Done)
5.2.4 Test Execution Schedule
5.2.5 Factors Influencing the Test Effort
5.2.6 Test Estimation Techniques

5.2.1 Purpose and Content of a Test Plan

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
- Selecting metrics for test monitoring and control

5.2.3 Entry Criteria and Exit Criteria (Definition of Ready and Definition of Done)

- Analytical
- Model-Based
- Methodical
- Process-compliant
- Directed
- Regression-averse
- Reactive

5.2.3 Entry Criteria and Exit Criteria (Definition of Ready and Definition of Done)

Entry Criteria

- 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

Exit criteria

- Planned tests have been executed
- 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

5.2.3 Entry Criteria and Exit Criteria (Definition of Ready and Definition of Done)

Entry Criteria

- Availability of necessary test tools
- Availability of test data and other necessary resources

exit criteria

- 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

5.2.4 Test Execution Schedule

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

5.2.5 Factors Influencing the Test Effort

- Product characteristics
- Development process characteristics
- People characteristics
- Test results

5.2.6 Test Estimation Techniques

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

5.3 Test Monitoring and Control

5.3.1 Metrics Used in Testing

5.3.2 Purposes, Contents, and Audiences for Test Reports

5.3.1 Metrics Used in Testing

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

5.3.2 Purposes, Contents, and Audiences for Test Reports

Typical 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
- Reusable test work products produced

5.4 Configuration Management

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

5.5 Risks and Testing

5.5.1 Definition of Risk
5.5.2 Product and Project Risks
5.5.3 Risk-based Testing and Product Quality

5.5.1 Definition of Risk

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.

5.5.2 Product and Project Risks

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.

- 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

Project risk involves situations that, should they occur, may have a negative effect on a project's ability to achieve its objectives

- Project issues
  - Delays may occur in delivery, task completion, or satisfaction of exit criteria or definition of done
- Organizational issues
  - Users, business staff, or subject matter experts may not be available due to conflicting business priorities

5.5.3 Product and Project Risks

- 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

- Political issues
  - There may be an improper attitude toward, or expectations of, testing (e.g., not appreciating the value of finding defects during testing)
- Technical issues
  - 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

5.5.3 Product and Project Risks

- User experience (UX) feedback might not meet product expectations
- Supplier issues
  - Contractual issues may cause problems to the project

5.5.4 Risk-based Testing and Product Quality

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

5.6 Defect Management

...

Since one of the objectives of testing is to find defects, defects found during testing should be logged

Defects may be reported during coding, static analysis, reviews, or during dynamic testing, or use of a software product. Defects may be reported for issues in code or working systems, or in any type of documentation including requirements, user stories and acceptance criteria, development documents, test documents, user manuals, or installation guides. In order to have an effective and efficient defect management process, organizations may define standards for the attributes, classification, and workflow of defects

Tool Support for Testing

6.1 Test Tool Considerations.

6.2 Effective Use of Tools.

6.1 Test Tool Considerations

6.1.1 Test Tool Classification

6.1.2 Benefits and Risks of Test Automation

6.1.3 Special Considerations for Test Execution and Test Management Tools

6.1.1 Test Tool Classification

Test Tool Classification

- Tool support for management of testing and testware
- Tool support for static testing
- Tool support for test design and implementation
- Tool support for test execution and logging
- Tool support for performance measurement and dynamic analysis
- Tool support for specialized testing needs

6.1.2 Benefits and Risks of 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

Potential risks of using tools to support testing include:

- Expectations for the tool may be unrealistic (including functionality and ease of use)
- A new platform or technology may not be supported by the tool
- An open source project may be suspended
- Version control of test work products may be neglected

6.1.2 Benefits and Risks of Test Automation

- 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)

6.1.3 Special Considerations for Test Execution and Test Management Tools

Test execution tools

- Capturing test approach
- Data-driven test approach
- Keyword-driven test approach

Test management tools

- 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

6.2.1 Main Principles for Tool Selection

6.2.2 Pilot Projects for Introducing a Tool into an Organization

6.2.3 Success Factors for Tools

6.2.1 Main Principles for Tool Selection

- Assessment of the maturity of the own organization, its strengths and weaknesses
- Understanding of the technologies used by the test object(s), in order to select a tool that is compatible with that technology
- 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)

6.2.2 Pilot Projects for Introducing a Tool into an Organization

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

6.2.2 Pilot Projects for Introducing a Tool into an Organization

- Deciding on standard ways of using, managing, storing, and maintaining the tool and the test work products (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

6.2.3 Success Factors for Tools

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)
- Providing support to the users of a given tool
- Gathering lessons learned from all users

References

Certified Tester Foundation Level (CTFL) Syllabus

Version 2018 v3.1.1

Thank you

Thank you very much for accompanying me in studying together in ISTQB with Aisyah, Ain, Aniyah, Ikha, Fachri, Mizan, and Hazli. By learning together, we can correct each other and understand the text and context that are present.

Find us on medium :

1. why testing necessary (zulaikha)
2. Risk and testing (syuhada)
3. test levels and test type (aniyah)

Including the previous QA team member ryan, charis, Farah, hana, mbak maya and mas taufik. and all of product team AVANA (Backend, Frontend, mobile, product design, product manager)