Quality Resources & Solutions

Tài liệu tóm tắt ôn thi chứng chỉ quốc tế ISTQB

Giảng viên: Tạ Thị Thinh

Email: thinhtt0204@gmail.com

SDT: 0986775464

Skype: ta.thinh0204

Bản quyền thuộc về QRS-website: qr-solutions.com.vn
DC: tầng 3, số 1, ngõ 36/7 Duy Tân

Bản quyền thuộc về QRS-website: qr-solutions.com.vn

Bản quyền thuộc về QRS-website: qr-solutions.com.vn
DC: tầng 3, số 1, ngõ 36/7 Duy Tân

CHAPTER 0: ISTQB introduction

Nội dung chính:

1. **ISTQB Foundation Level Examination**

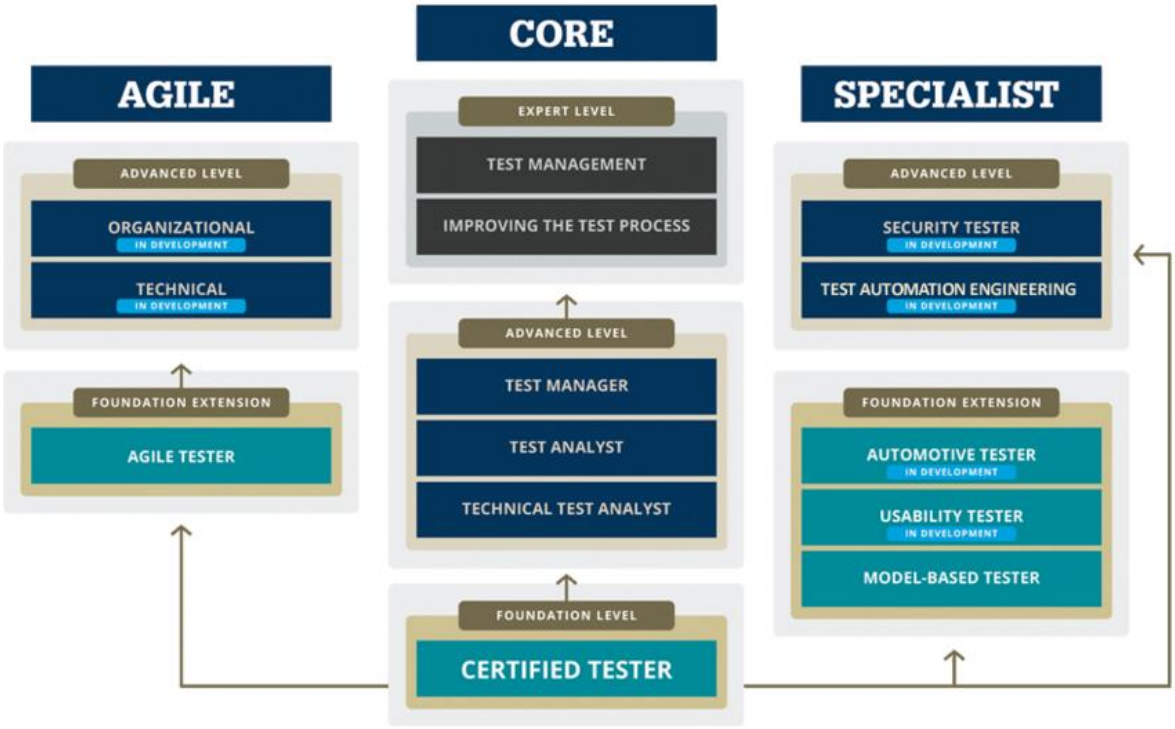
- Giới thiệu về bài thi ISTQB
- Who is ISTQB -ISTQB là gì?
- Course structure -Exams in ISTQB
- About Foundation level Examination -Về các mức độ kiến thức trong bài thi

2. **How to study ISTQB and prepare for exam**

- Plan for examination- Lập kế hoạch cho kỳ thi
- How to read and use ISTQB books- Cách đọc và sử dụng các cuốn sách ISTQB bằng tiếng Anh
- Tips for doing exercises-Các chỉ dẫn, kinh nghiệm để làm bài tập

1. ISTQB Foundation Level Examination

1.1 ISTQB Course structure (Cấu trúc ISTQB)



- There are currently three levels of certification

Có 3 mức độ của chứng chỉ

  - The Foundation Level certification (one module) - CTFL (Certified Tester Foundation level

Chứng chỉ ở mức cơ bản (1 mảng)

  - Advanced Level certifications (three modules)

Chứng chỉ ở cao hơn (3 mảng: quản lý, kỹ thuật test và automation)

  - An Expert levels is currently being developed

Chứng chỉ ở mức chuyên gia đang được xây dựng

- For all three levels, international working groups develop and maintain internationally uniform curricula and exams - Đối với tất cả 3 mức chứng chỉ, thì đều thống nhất tài liệu giảng và bài thi trên toàn thế giới.

Bản quyền thuộc về QRS-website: qr-solutions.com.vn

1.2 Who is ISTQB (ISTQB là gì)?

- ISTQB là 1 tổ chức phi lợi nhuận có trách nhiệm định nghĩa ra các hướng dẫn như cấu trúc bài thi, quy ước, khái niệm, chứng chỉ,..
- Nhóm làm việc trong tổ chức ISTQB có trách nhiệm phát triển và duy trì các giáo trình và xây dựng bài thi chung trên toàn thế giới.

1.3 Exams in ISTQB (Bài thi trong ISTQB cập nhật phiên bản 2018)

2. How to study ISTQB and prepare for exam

Cách nghiên cứu và chuẩn bị cho kỳ thi lấy chứng chỉ ISTQB

2.1 How to read and use ISTQB books

Cách đọc và sử dụng các cuốn sách ISTQB bằng tiếng Anh

Sách dùng để thi chính thức ISTQB gồm có:

1. ISTQB FOUNDATIONS LEVEL
Giới thiệu và giải thích 1 cách đầy đủ về các khái niệm và hoạt động của testing. Gồm có 207 trang.
2. ISTQB FOUNDATION LEVEL SYLLABUS
Tóm tắt các khái niệm chính về testing, những key word cần nhớ của quyển 1
3. ISTQB GLOSSARY OF TESTING TERMS
Từ điển các khái niệm về testing

Ngoài ra các bộ đề thi, các ví dụ về mẫu bài thi có rất nhiều trên mạng

Mục tiêu nghiên cứu các tài liệu ISTQB:

- K1 Remember – Ghi nhớ
- K2 Understand – Hiểu biết
- K3 Apply – Áp dụng

2.2 Foundation Level examination – Bài thi ở mức cơ sở

1. Cấu trúc bài thi
   - Thi trắc nghiệm multi choices với 40 câu hỏi
   - Mỗi câu trả lời đúng là 1 điểm, tối đa 40 điểm
   - Trả lời đúng 65%, tức là 26 câu thì mới PASS
   - Thời gian làm bài là 60 phút cho thí sinh nói tiếng Anh là tiếng mẹ đẻ, thí sinh ở vùng khác là 75 phút
2. Phân bổ câu hỏi của đề thi
   Điểm khác biệt so với đề thi cũ là chỉ dùng 3 khái niệm K1- Remember, K2- Understand, K3- Analysis và số câu hỏi thuộc K2 tăng lên nhiều hơn so với đề cũ
   - K1 – 8 câu hỏi
   - K2 – 24 câu hỏi
   - K3 – 8 câu hỏi
3. Phân bổ theo chương
   Bài thi vẫn dựa chủ yếu vào sách Syllabus 2018 để ra đề thi và phân bố theo chương như sau

Chương	Tổng số câu	Phân bổ theo độ khó

Chương 1- Fundamental of testing	8 câu	K1 = 2 K2 = 6 K3 = 0
Chương 2- Test throughout lifecycles	5 câu	K1 = 1 K2 = 4 K3 = 0
Chương 3- Static testing	5 câu	K1 = 1 K2 = 3 K3 = 1
Chương 4- Test Design Techniques	11 câu	K1 = 1 K2 = 5 K3 = 5
Chương 5- Test management	9 câu	K1 = 2 K2 = 5 K3 = 2
Chương 6- Test tools	2 câu	K1 = 1 K2 = 1 K3 = 0

2.3 Plan for examination -Lập kế hoạch cho kỳ thi

- Chuẩn bị thi trong vòng 1 tháng đến 1.5 tháng
- Mỗi tuần tối thiểu phải đọc hết 1 chương và làm bài tập

Tips for doing exercises -Các chỉ dẫn, kinh nghiệm để làm bài tập

- Bạn phải nắm chắc kiến thức, thậm chí có thể phải thuộc lý thuyết trong sách
- Bạn nên tạo bản tóm tắt bằng tiếng Anh ngắn gọn các kiến thức này để việc ghi nhớ dễ dàng hơn
- Bạn cần tăng cường đọc đi đọc lại sách syllabus nhiều lần để tăng khả năng đọc hiểu tiếng Anh của mình
- Cần có sự phân bố thời gian hợp lý cho từng phần để ôn tập tốt hơn. Ví dụ:
    - Chương 4 có nhiều bài tập để làm, nhưng đi thi cũng chỉ có 3 câu bài tập còn lại là lý thuyết. Vì thế việc luyện nhiều bài tập không giúp được gì nhiều
    - Chương 5 là chương khó hiểu nhất trong các nội dung và thường mất nhiều điểm ở đây.

Cuối cùng, bạn hãy chuẩn bị tốt cho kỳ thi, khi đã lên mục tiêu thì hãy tập trung học trong thời gian ngắn, thường việc ôn thi chỉ mất từ 1-2 tháng là được. Việc kéo dài 6 tháng ôn thi cũng không giúp được gì nhiều lắm.

Chapter 1: Fundamentals of Testing

1.1 What is Testing? - Testing là gì

What is testing?

- Test activities exist before and after test execution.
- Both dynamic testing and static testing

1.1.1 Typical Objectives of Testing

- To evaluate work products such as requirements, user stories, design, and code
- To verify whether all specified requirements have been fulfilled
- To validate whether the test object is complete and works as the users and other stakeholders expect
- To build confidence in the level of quality of the test object
- To prevent defects
- To find failures and defects
- To provide sufficient information to stakeholders to allow them to make informed decisions
- To reduce the level of risk of inadequate software quality
- To comply with contractual, legal, or regulatory requirements or standards

Different viewpoints in testing take different objectives into account

- During component testing, one objective may be to find as many failures as possible so that the underlying defects are identified and fixed early. Another objective may be to increase code coverage of the component tests.
- During acceptance testing, one objective may be to confirm that the system works as expected and satisfies requirements. Another objective of this testing may be to give information to stakeholders about the risk of releasing the system at a given time.

1.1.2 Testing and Debugging

Testing and debugging are different.

- Executing tests can show failures that are caused by defects in the software.
- Debugging is the development activity that finds, analyzes, and fixes such defects.
- Subsequent confirmation testing checks whether the fixes resolved the defects.

1.2 Why is Testing Necessary?

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

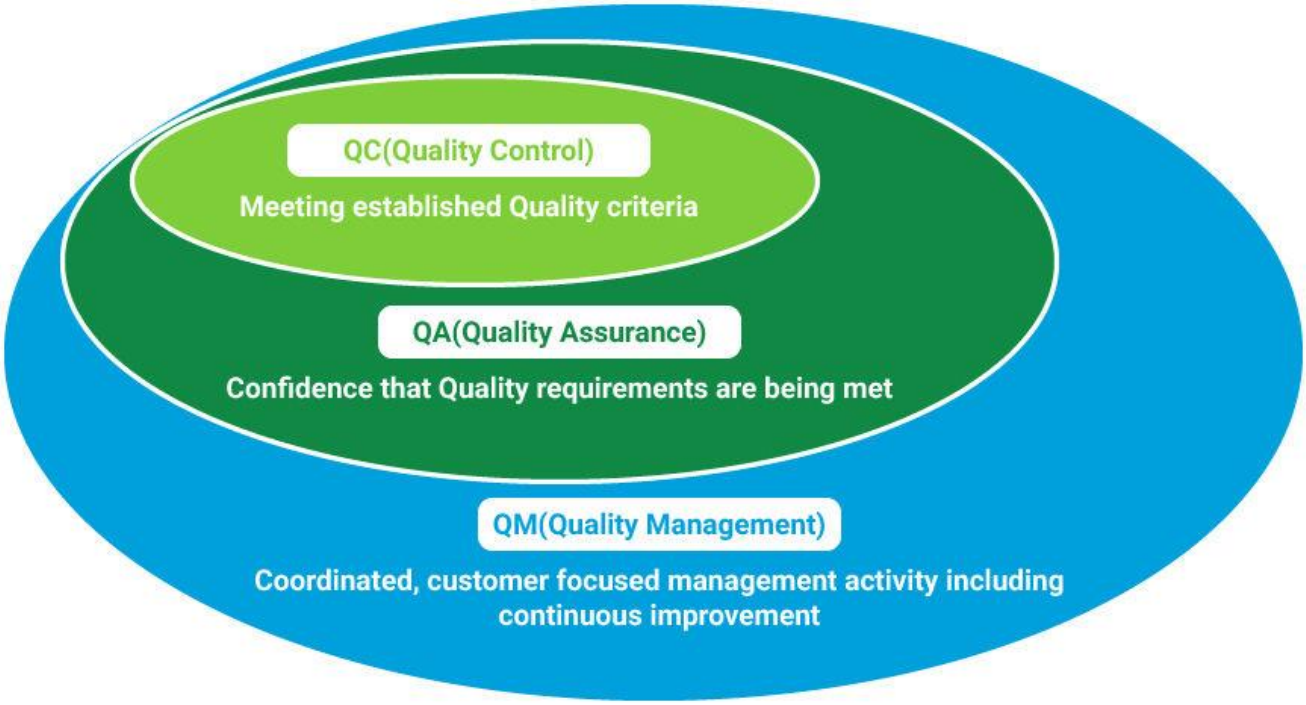
1.2.1 Testing's Contributions to Success

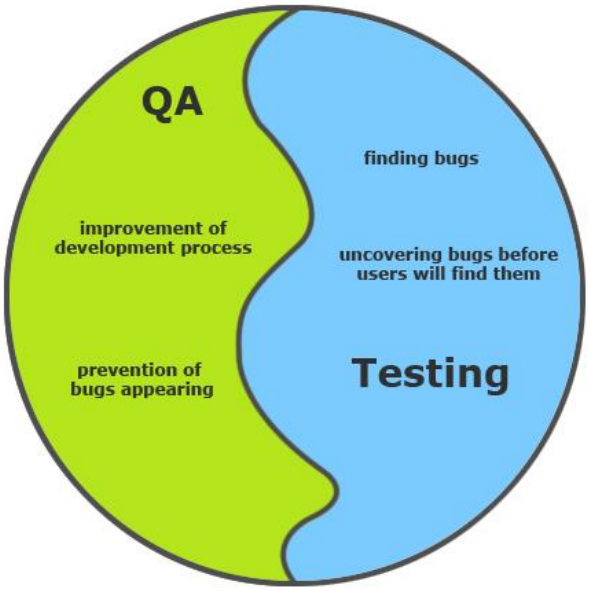
Testing contributes to overall software development and maintenance success.
Example:

Phase	Contributes

Testers involved in requirements reviews or user story refinement could detect defects	Reduces the risk of incorrect or untestable functionality being developed.
Testers work closely with system designers while the system is being designed can increase understanding and how to test it	Reduce the risk of fundamental design defects and enable tests to be identified at an early stage.
Testers work closely with developers while the code can increase understanding and how to test it	Reduce the risk of defects within the code and the tests.
Testers verify and validate the software prior to release can detect failures	Increases the likelihood that the software meets stakeholder needs and satisfies requirements.

1.2.2 Quality Assurance and Testing





Quality assurance and testing are not the same, but they are related

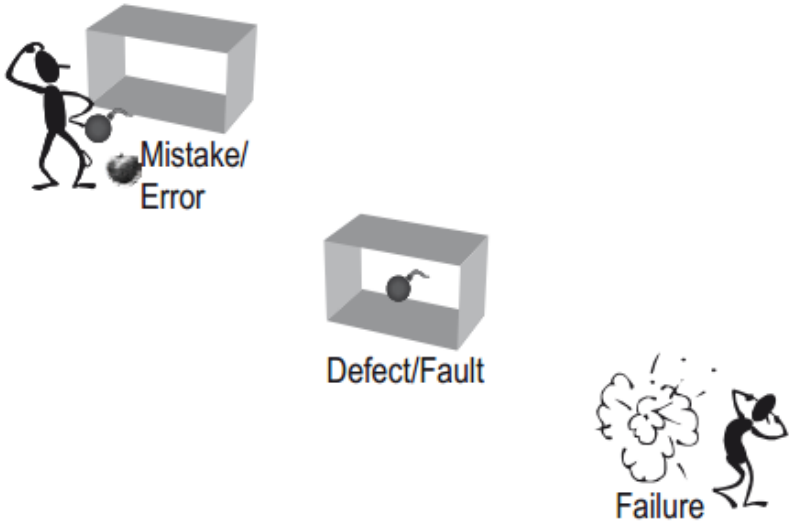
- Quality management includes all activities that direct and control an organization with regard to quality, includes both quality assurance and quality control
- Quality assurance is typically focused on adherence to proper processes, in order to provide confidence that the appropriate levels of quality will be achieved. Quality assurance contributes to defect prevention (example: the use of root cause analysis to detect and remove the causes of defects, retrospective meetings to improve processes)
- 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

1.2.3 Errors, Defects, and Failures

Mistake: Human action that produces an incorrect result.

Defect: Manifestation of an error in software.

Failure: Deviation of the software from its expected delivery or service.



Errors may occur for many reasons, such as:

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

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.

1.2.4 Defects, Root Causes and Effects

The root causes of defects are the earliest actions or conditions that contributed to creating the defects - Identify the root causes of failure can

- Reduce the occurrence of similar defects in the future
- Lead to process improvements that prevent a significant number of future defects from being introduced

1.3 Seven Principles of Testing – 7 nguyên lý cơ bản của testing

Testing shows presence of defects

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

Exhaustive testing is impossible – Complete test

- Test everything (all combination of inputs and preconditions) is not feasible
- Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts

Early testing

- Testing activities should start as early as possible in the software or software development life cycle and should focus on defined objectives
- Perform the test design and review activities early can find defects early on when they are cheap to find and fix.

Defect clustering: defect density

- A small numbers of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures
- Rule 80/20: Module core often contains 80% defects

Pesticide paradox

- If the same tests are repeated over and over again, no new defects can be found.
- To overcome this pesticide paradox, test cases need to be regularly reviewed and revised; new and different tests need to be written to exercise different parts of the software to find potentially more defects

Testing is context dependent

- Testing is done differently in different context
- Ex: Safety-critical software is tested differently from an ecommerce site

Absence of error fallacy

- 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

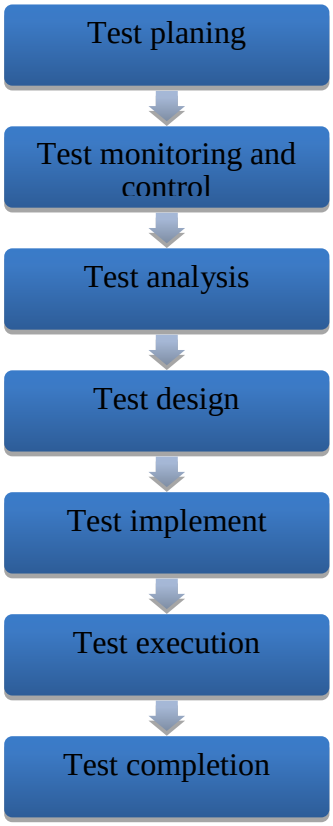
1.4 Test Process

1.4.1 Test Process in Context

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



In Agile development involves small iterations of software design, build, and test that happen on a continuous basis, supported by on-going planning. So test activities are also happening on an iterative, continuous basis within this development approach.

In sequential development, the stepped logical sequence of activities will involve overlap, combination, concurrency, or omission, so tailoring these main activities within the context of the system and the project is usually required.

Test activities	Objectives	Work products
Test planning	- Define the objectives of testing and the approach for meeting test objectives (e.g., specifying suitable test techniques and tasks, and formulating a test schedule for meeting a deadline). - May be revisited based on feedback from monitoring and control activities	One or more test plans includes: - information about the test basis - to which the other test work products will be related via traceability information - exit criteria (or definition of done) which will be used during test monitoring and control
Test monitoring and control	- Involves the on-going comparison of actual progress against the test plan using any test monitoring metrics defined in the test plan - Involves taking actions necessary to meet the objectives of the test plan - Are supported by the evaluation of exit criteria for test execution as part of a	test progress reports (produced on an ongoing and/or a regular basis) test summary reports (produced at various completion milestones)

	given test level may include: Checking test results and logs against specified coverage criteria + Assessing the level of component or system quality based on test results and logs + Determining if more tests are needed - Provide test progress reports, including deviations from the plan and information to support any decision to stop testing	
Test analysis	Analyzing the test basis appropriate to the test level being considered: - Requirement specifications - Design and implementation information - The implementation of the component or system - Risk analysis reports Evaluating the test basis and test items to identify defects of various types, such as: - Ambiguities - Omissions - Inconsistencies - Inaccuracies - Contradictions - Superfluous statements Identifying features and sets of features to be tested - Defining and prioritizing test conditions (functional, non-functional, and structural characteristics, other business and technical factors, and levels of risks) - Capturing bi-directional traceability between each element of the test basis and the associated test conditions	defined and prioritized test conditions bi-directionally traceable test charters reporting of defects in the test basis
Test design	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	High-level test cases, without concrete values for input data and expected results Bi-directionally traceable to the test condition(s) it covers. Test data, the design of the test environment, and the identification of infrastructure and tools, though

	tools Capturing bi-directional traceability between the test basis, test conditions, test cases, and test procedures	the extent to which these results are documented varies significantly. Test conditions defined in test analysis may be further refined in test design
Test implementation	Developing and prioritizing test rocedures, and, potentially, creating automated test scripts Creating test suites from the test rocedures and (if any) 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 procedures and the sequencing of those test procedures Test suites A test execution schedule The test data serve to assign concrete values to the inputs and expected results of test cases The concrete expected results which are associated with concrete test data are identified by using a test oracle.
Test execution	Recording the IDs and versions of the test item(s) or test object, test tool(s), and testware Executing tests either manually or by using test execution tools Comparing actual results with expected results Analyzing anomalies to establish their likely causes Reporting defects based on the failures observed Logging the outcome of test execution (e.g., pass, fail, blocked) Repeating test activities either as a result of action taken for an anomaly, or as part of the planned testing Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test results.	Documentation of the status of individual test cases or test procedures (e.g., ready to run, pass, fail, blocked, deliberately skipped, etc.) Defect reports Documentation about which test item(s), test object(s), test tools, and testware were involved in the testing
Test completion	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	Test summary reports Action items for improvement of subsequent projects or iterations Change requests or product backlog

	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	items Finalized testware.

1.4.4 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

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

Identifying defects may be perceived as criticism of the product and of its author. An element of human psychology called confirmation bias can make it difficult to accept information that disagrees with currently held beliefs.

Some people may perceive testing as a destructive activity, even though it contributes greatly to project progress and product quality

To reduce these perceptions:

- Information about defects and failures should be communicated in a constructive way
- Good interpersonal skills to be able to communicate effectively about defects, failures, test results, test progress, and risks, and to build positive relationships with colleagues
- Good communicate:
    - Start with collaboration rather than battles. Remind everyone of the common goal of better quality systems.
    - Emphasize the benefits of testing.
    - Communicate test results and other findings in a neutral, fact-focused way without criticizing the person who created the defective item.
    - 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.
- Clearly defining the right set of test objectives has important psychological implications

1.5.2 Tester's and Developer's Mindsets

Developers and testers often think differently

	Developer	Tester
objective	design and build a product	verifying and validating the product, finding defects prior to release
mindset	more interested in designing and building solutions than in contemplating what might be wrong with those solutions difficult to find mistakes in their own work developers should be able to test their own code	curiosity, professional pessimism, a critical eye, attention to detail, and a motivation for good and positive communications and relationships

Some of the test activities done by independent testers:

- to increases defect detection effectiveness, which is particularly important for large, complex, or safety-critical systems.
- they have different cognitive biases from the authors

Chapter 2: Testing Throughout the Software Development Lifecycle

2.1 Software Development Lifecycle Models

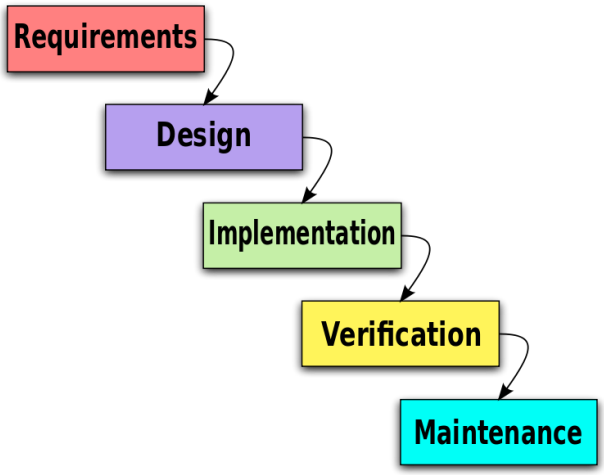
In any software development lifecycle model, there are several characteristics of good testing:

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

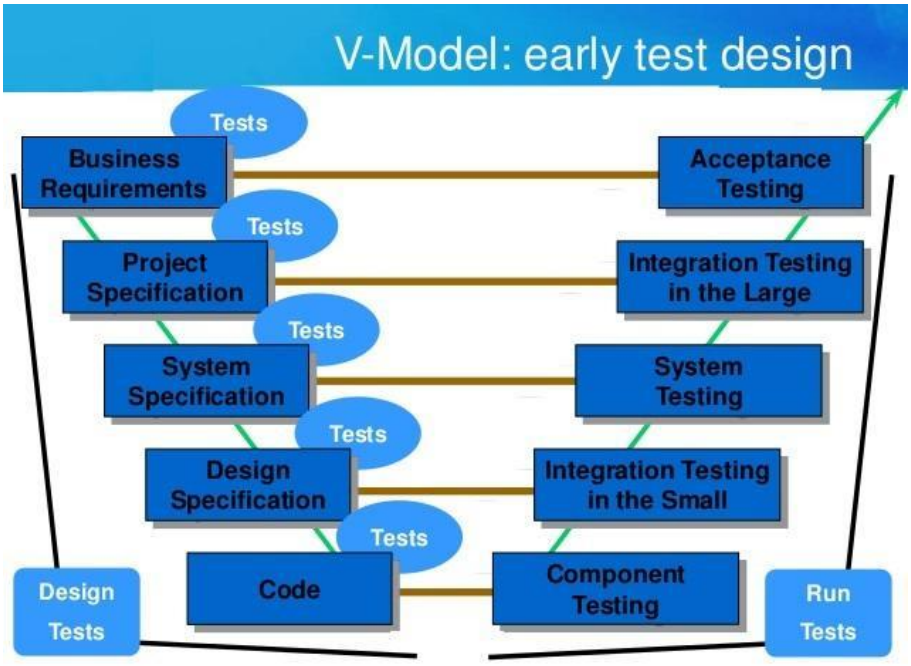
In any model, test activities should start in the early stages of the lifecycle, adhering to the testing principle of early testing. There are two common models:

- Sequential development models
- Iterative and incremental development models

1. Sequential development models



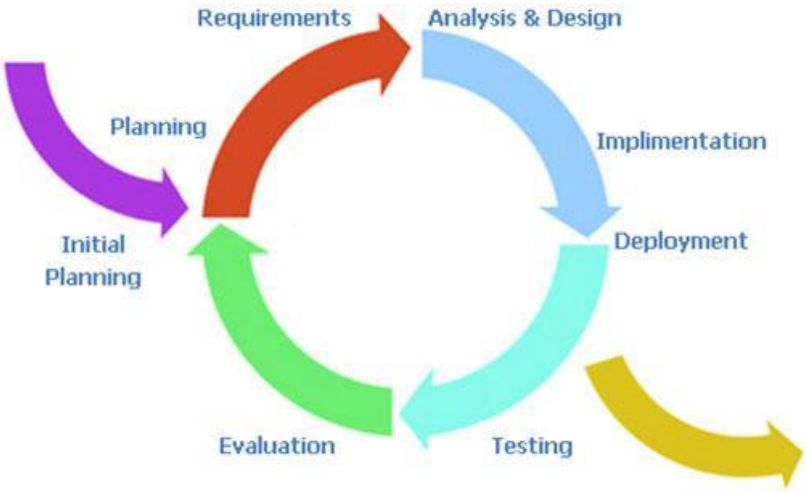
In the Waterfall model, the development activities (e.g., requirements analysis, design, coding, testing) are completed one after another. In this model, test activities only occur after all other development activities have been completed.



The V-model integrates the test process throughout the development process, implementing the principle of early testing. Further, the V-model includes test levels associated with each corresponding development phase, which further supports early testing. In this model, the execution of tests associated with each test level proceeds sequentially, but in some cases overlapping occurs

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

2. Iterative and incremental development models



Examples of Incremental development:

- Rational Unified Process: long iterations (e.g., two to three months), and the feature increments are correspondingly large, such as two or three groups of related features
- Scrum: short iterations (e.g., hours, days, or a few weeks), and the feature increments are correspondingly small, such as a few enhancements and/or two or three new features

- Kanban: Implemented with or without fixed-length iterations, which can deliver either a single enhancement or feature upon completion, or can group features together to release at once
- Spiral (or prototyping): Involves creating experimental increments, some of which may be heavily re-worked or even abandoned in subsequent development work

Incremental development has some characteristics:

- Establishing requirements, designing, building, and testing a system in pieces, which means that the software's features grow incrementally.
- Groups of features are specified, designed, built, and tested together in a series of cycles, often of a fixed duration.
- Involve changes to features developed in earlier iterations, along with changes in project scope.
- Each feature is tested at several test levels as it moves towards delivery.
- In some cases, teams use continuous delivery or continuous deployment, both of which involve significant automation of multiple test levels as part of their delivery pipelines.
- Release to end-users on a feature-by-feature basis, on an iteration- by-iteration basis, or in a more traditional major-release fashion.
- Regression testing is increasingly important as the system grows.

2.1.2 Software Development Lifecycle Models in Context

Software development lifecycle models must be selected and adapted to the context of project and product characteristics:

- the project goal
- the type of product being developed
- business priorities (e.g., time-to-market)
- 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

Software development lifecycle models themselves may be combined. For example, a V-model may be used for the development and testing of the backend systems and their integrations, while an Agile development model may be used to develop and test the front-end user interface (UI) and functionality. Prototyping may be used early in a project, with an incremental development model adopted once the experimental phase is complete.

2.2 Test Levels (K2)

The test levels:

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

Test levels are characterized by the following attributes:

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

Common objectives of test levels:

- Reducing risk
- Verifying whether the functional and non-functional behaviors
- Building confidence that changes have not broken existing components, systems
- Finding defects
- Preventing defects from escaping to higher test levels

Item	**Component testing**	**Integration Testing**	**System Testing**
Objective	Focuses on components that are separately testable	focuses on interactions and interfaces between components or systems	focuses on the behavior and capabilities of a whole system or product
Special	Done in isolation from the rest of the system, require mock objects, service virtualization, harnesses, stubs, and drivers.	two different levels of integration testing: Component integration testing System integration testing	Incompleted or undocumented specification
Test types	Functionality (e.g., correctness of calculations) Non-functional characteristics (e.g., searching for memory leaks) Structural properties (e.g., decision testing).	Functional and non-functional	Functional and non-functional and data quality characteristic
Test basis	Detailed design Code Data model Component specifications Test objects	Software and system design Sequence diagrams Interface and communication protocol specifications Use cases Architecture at component or system level Workflows External interface definitions	System and software requirement specifications (functional and non-functional) Risk analysis reports Use cases Epics and user stories Models of system behavior State diagrams System and user manuals
Typical test objects	Components, units or modules Code and data structures	Subsystems Databases Infrastructure	Applications Hardware/software systems

	Classes Database modules	Interfaces APIs Microservices	Operating systems System under test (SUT) System configuration and configuration data
Environment	Development environment with framework, debug tool,...	Specific environment	Correspond to the production environment
Typical defects and failures	Incorrect functionality (e.g., not as described in design specifications) Data flow problems Incorrect code and logic Defects are typically fixed as soon as they are found	Inconsistent message structures between systems 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 component Failure to comply with mandatory security regulations	Incorrect calculations Incorrect or unexpected system functional or non-functional behavior Incorrect control and/or data flows within the system Failure to properly and completely carry out end-to-end functional tasks Failure of the system to work properly in the production environment(s) Failure of the system to work as described in system and user manuals
Responsibilities	Developer	Developer and tester	Independent testers
Approach	Test-first approach Test driven development (TDD)	The greater the scope of integration, the more difficul becomes to isolate defects to specific component or systen Big-bang integration Incremental integration	The most appropriate techniques for the aspect(s) of the system to be tested

2.2.4 Acceptance Testing

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

Common forms of acceptance testing:

1. **User acceptance testing (UAT)**

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

2. **Operational acceptance testing (OAT)**

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

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

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

3. **Contractual and regulatory acceptance testing**

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

Regulatory acceptance testing is performed against any regulations that must be adhered to, such as government, legal, or safety regulations

Contractual acceptance testing and regulatory acceptance testing are often performed by users or by independent testers

4. **Alpha and beta testing**

Alpha and beta testing are typically used by developers of commercial off-the-shelf (COTS) software who want to get feedback from potential or existing users, customers, and/or operators before the software product is put on the market. Alpha testing is performed at the developing organization's site, not by the development team, but by potential or existing customers, and/or operators or an independent test team.

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

Test basis

- Business processes
- User or business requirements
- Regulations, legal contracts and standards
- Use cases
- System requirements
- System or user documentation
- Installation procedures
- Risk analysis reports

In addition, as a test basis for deriving test cases for operational acceptance testing, one or more of the following work products can be used:

- Backup and restore procedures
- Disaster recovery procedures
- Non-functional requirements
- Operations documentation
- Deployment and installation instructions
- Performance targets
- Database packages
- Security standards or regulations

Typical test objects

- 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

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 may also occur at other times, for example:

In a sequential development lifecycle

- Acceptance testing of a COTS software product may occur when it is installed or integrated
- Acceptance testing of a new functional enhancement may occur before system testing

In iterative development

- Acceptance testing during and at the end of each iteration, it focused on verifying a new feature against its acceptance criteria and those focused on validating that a new feature satisfies the users' needs
- Alpha tests and beta tests may occur, either at the end of each iteration, after the completion of each iteration, or after a series of iterations
- User acceptance tests, operational acceptance tests, regulatory acceptance tests, and contractual acceptance tests also may occur, either at the close of each iteration, after the completion of each iteration, or after a series of iterations

2.3 Test Types

- Evaluating functional quality characteristics, such as completeness, correctness, and appropriateness
- Evaluating non-functional quality characteristics, such as reliability, performance efficiency, security, compatibility, and usability
- Evaluating whether the structure or architecture of the component or system is correct, complete, and as specified
- Evaluating the effects of changes, such as confirming that defects have been fixed (confirmation testing) and looking for unintended changes in behavior resulting from software or environment changes (regression testing)

- ISO standard (ISO/IEC 25010)

Characteristic	Sub-Characteristics
Functionality	Accuracy, suitability, interoperability, compliance
	Security
Reliability	Maturity (robustness), fault-tolerance, recoverability, compliance
Usability	Understandability, learnability, operability, attractiveness, compliance
Efficiency	Performance (time behavior), resource utilization, compliance
Maintainability	Analyzability, changeability, stability, testability, compliance
Portability	Adaptability, installability, co-existence, replaceability, compliance

2.3.1 Functional Testing

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

2.3.2 Non-functional Testing

Non-functional testing of a system evaluates characteristics of systems and software such as usability, performance efficiency or security

2.3.3 White-box Testing

White-box testing derives tests based on the system's internal structure or implementation

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.

2.3.4 Change-related Testing

- Confirmation testing: confirm whether the original defect has been successfully fixed.
- Regression testing: a change made in one part of the code, may accidentally affect the behavior of other parts of the code. Such unintended side-effects are called regressions. Regression testing involves running tests to detect such unintended side-effects. Changes may include changes to the environment, such as a new version of an operating system or database management system.

2.3.5 Test Types and Test Levels

Examples of functional, non-functional, white-box, and change-related tests will be given across all test levels, for a banking application

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

2.4 Maintenance Testing

Maintenance testing focuses on testing the changes to the system, as well as testing unchanged parts that might have been affected by the changes. 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. The scope of maintenance testing depends on:

- The degree of risk of the change, for example, the degree to which the changed area of software communicates with other components or systems
- The size of the existing system
- The size of the change

2.4.1 Triggers for Maintenance

We can classify the triggers for maintenance as follows:

- Modification, such as planned enhancements (e.g., release-based), corrective and emergency changes, changes of the operational environment (such as planned operating system or database upgrades), upgrades of COTS software, and patches for defects and vulnerabilities
- Migration, such as from one platform to another, which can require operational tests of the new environment as well as of the changed software, or tests of data conversion when data from another application will be migrated into the system being maintained
- Retirement, such as when an application reaches the end of its life

2.4.2 Impact Analysis for Maintenance

Impact analysis evaluates the changes that were made for a maintenance release to identify the intended consequences as well as expected and possible side effects of a change, and to identify the areas in the system that will be affected by the change.

Impact analysis can also help to identify the impact of a change on existing tests. The side effects and affected areas in the system need to be tested for regressions, possibly after updating any existing tests affected by the change.

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

Impact analysis can be difficult if:

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

CHAPTER 3 STATIC TESTING

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

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)

Reviews can be applied to any work product that the participants know how to read and understand.

Static analysis can be applied efficiently to any work product with a formal structure (typically code or models) for which an appropriate static analysis tool exists.

Static analysis can even be applied with tools that evaluate work products written in natural language such as requirements (e.g., checking for spelling, grammar, and readability).

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

Compare with dynamic testing, static testing can:

- Provide an assessment of the quality of the work products
- Identify defects as early as possible
- Finds defects in work products directly rather than identifying failures caused by defects when the software is run.
- Find the defect with much less effort
- Used to improve the consistency and internal quality of work products

Typical defects that are easier and cheaper to find and fix through static testing include:

- Requirement defects (e.g., inconsistencies, ambiguities, contradictions, omissions, inaccuracies, and redundancies)
- Design defects (e.g., inefficient algorithms or database structures, high coupling, low cohesion)
- Coding defects (e.g., variables with undefined values, variables that are declared but never used, unreachable code, duplicate code)
- Deviations from standards (e.g., lack of adherence to coding standards)
- Incorrect interface specifications (e.g., different units of measurement used by the calling system than by the called system)

- Security vulnerabilities (e.g., susceptibility to buffer overflows)
- Gaps or inaccuracies in test basis traceability or coverage (e.g., missing tests for an acceptance criterion)
- Maintainability defects (e.g., improper modularization, poor reusability of components, code that is difficult to analyze and modify without introducing new defects)

3.2 Review Process

3.2.1 Work Product Review Process

Planning

- Defining the scope, which includes the purpose of the review, what documents or parts of documents to review, and the quality characteristics to be evaluated
- Estimating effort and timeframe
- Identifying review characteristics such as the review type with roles, activities, and checklists
- Selecting the people to participate in the review and allocating roles
- Defining the entry and exit criteria for more formal review types (e.g., inspections)
- Checking that entry criteria are met (for more formal review types)

Initiate review

- Distributing the work product (physically or by electronic means) and other material, such as issue log forms, checklists, and related work products
- Explaining the scope, objectives, process, roles, and work products to the participants
- Answering any questions that participants may have about the review

Individual review (i.e., individual preparation)

- Reviewing all or part of the work product
- Noting potential defects, recommendations, and questions

Issue communication and analysis

- Communicating identified potential defects (e.g., in a review meeting)
- Analyzing potential defects, assigning ownership and status to them
- Evaluating and documenting quality characteristics
- Evaluating the review findings against the exit criteria to make a review decision (reject; major changes needed; accept, possibly with minor changes)

Fixing and reporting

- Creating defect reports for those findings that require changes
- Fixing defects found (typically done by the author) in the work product reviewed
- Communicating defects to the appropriate person or team (when found in a work product related to the work product reviewed)
- Recording updated status of defects (in formal reviews), potentially including the agreement of the comment originator
- Gathering metrics (for more formal review types)
- Checking that exit criteria are met (for more formal review types)
- Accepting the work product when the exit criteria are reached

3.2.2 Roles and responsibilities in a formal review

Author

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

Management

- Is responsible for review planning
- Decides on the execution of reviews
- Assigns staff, budget, and time
- Monitors ongoing cost-effectiveness
- Executes control decisions in the event of inadequate outcomes

Facilitator (often called moderator)

- Ensures effective running of review meetings (when held) Mediates, if necessary, between the various points of view
- Is often the person upon whom the success of the review depends?

Review leader

- Takes overall responsibility for the review
- Decides who will be involved and organizes when and where it will take place

Reviewers

- May be subject matter experts, persons working on the project, stakeholders with an interest in the work product, and/or individuals with specific technical or business backgrounds
- Identify potential defects in the work product under review
- May represent different perspectives (e.g., tester, programmer, user, operator, business analyst, usability expert, etc.)

Scribe (or recorder)

- Collates potential defects found during the individual review activity
- Records new potential defects, open points, and decisions from the review meeting (when held)

3.2.3 Review Types

Informal review (e.g., buddy check, pairing, pair review)

- Main purpose: detecting potential defects
- Possible additional purposes: generating new ideas or solutions, quickly solving minor problems
- Not based on a formal (documented) process
- May not involve a review meeting
- May be performed by a colleague of the author (buddy check) or by more people
- Results may be documented
- Varies in usefulness depending on the reviewers
- Use of checklists is optional
- Very commonly used in Agile development

Walkthrough

- Main purposes: 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 of the work product
- Scribe is mandatory
- Use of checklists is optional
- May take the form of scenarios, dry runs, or simulations
- Potential defect logs and review reports may be produced
- May vary in practice from quite informal to very formal

Technical review

- Main purposes: 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, considering alternative implementations
- Reviewers should be technical peers of the author, and technical experts in the same or other disciplines
- Individual preparation before the review meeting is required
- Review meeting is optional, ideally led by a trained facilitator (typically not the author)
- Scribe is mandatory, ideally not the author
- Use of checklists is optional
- Potential defect logs and review reports are typically produced

Inspection

- Main purposes: 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
- 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 (not the author)
- 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

3.2.4 Applying Review Techniques

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.

This technique is highly dependent on reviewer skills and may lead to many duplicate issues being reported by different reviewers.

Checklist-based

A checklist-based review is a systematic technique, whereby the reviewers detect issues based on checklists that are distributed at review initiation (e.g., by the facilitator).

A review checklist consists of a set of questions based on potential defects, which may be derived from experience.

Checklists should be specific to the type of work product under review and should be maintained regularly to cover issue types missed in previous reviews.

The main advantage of the checklist-based technique is a systematic coverage of typical defect types. Care should be taken not to simply follow the checklist in individual reviewing, but also to look for defects outside the checklist.

Scenarios and dry runs

In a scenario-based review, reviewers are provided with structured guidelines on how to read through the work product.

A scenario-based approach supports reviewers in performing “dry runs” on the work product based on expected usage of the work product (if the work product is documented in a suitable format such as use cases).

These scenarios provide reviewers with better guidelines on how to identify specific defect types than simple checklist entries.

As with checklist-based reviews, in order not to miss other defect types (e.g., missing features), reviewers should not be constrained to the documented scenarios.

Role-based

A role-based review is a technique in which the reviewers evaluate the work product from the perspective of individual stakeholder roles.

Typical roles include specific end user types (experienced, inexperienced, senior, child, etc.), and specific roles in the organization (user administrator, system administrator, performance tester, etc.).

Perspective-based

In perspective-based reading, similar to a role-based review, reviewers take on different stakeholder viewpoints in individual reviewing.

Typical stakeholder viewpoints include end user, marketing, designer, tester, or operations.

Using different stakeholder viewpoints leads to more depth in individual reviewing with less duplication of issues across reviewers.

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

Further, in perspective-based reading, checklists are expected to be used.

3.2.5 Success Factors for Reviews

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

People-related success factors for reviews include:

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

CHAPTER 4: Test Design Techniques

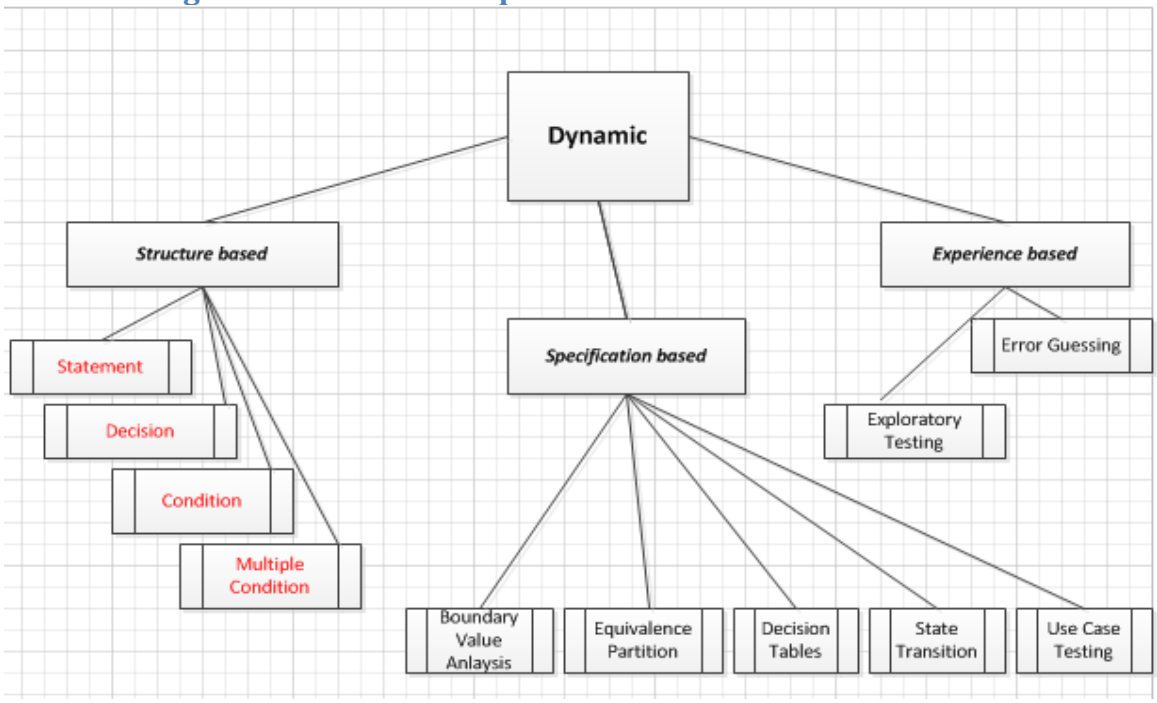
4.1 Categories of Test Techniques

4.1.1 Choosing Test Techniques

Depends on a number of factors:

- Type of component or system
- Component or system complexity
- Regulatory standards
- Customer or contractual requirements
- Risk levels
- Risk types
- Test objectives
- Available documentation
- Tester knowledge and skills
- Available tools
- Time and budget
- Software development lifecycle model
- Expected use of the software
- Previous experience with using the test techniques on the component or system to be tested
- The types of defects expected in the component or system

4.1.2 Categories of Test Techniques and Their Characteristics



Common characteristics of **specification-based test design** techniques include - Đặc điểm chung của kỹ thuật thiết kế test hướng đặc tả:

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

- Test cases may be used to detect gaps between the requirements and the implementation of the requirements, as well as deviations from the requirements
- Coverage is measured based on the items tested in the test basis and the technique applied to the test basis

Common characteristics of **structure-based test design** techniques include - Đặc điểm chung của kỹ thuật thiết kế test hướng cấu trúc:

- Test conditions, test cases, and test data are derived from a test basis that may include code, software architecture, detailed design, or any other source of information regarding the structure of the software
- Coverage is measured based on the items tested within a selected structure (e.g., the code or interfaces)
- Specifications are often used as an additional source of information to determine the expected outcome of test cases

Common characteristics of **experience-based test design** techniques include - Đặc điểm chung của kỹ thuật thiết kế test hướng kinh nghiệm:

- Test conditions, test cases, and test data are derived from a test basis that may include knowledge and experience of testers, developers, users and other stakeholders
- These techniques can be helpful in identifying tests that were not easily identified by other more systematic techniques. Depending on the tester's approach and experience, these techniques may achieve widely varying degrees of coverage and effectiveness.
- Coverage can be difficult to assess and may not be measurable with these techniques

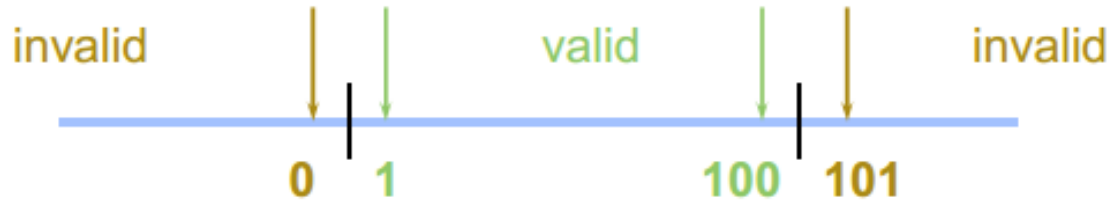
4.2 Black-box Test Techniques

4.2.1 Equivalence partitioning (EP) - Phân vùng tương đương (EP)

- Divide (partition) the inputs, outputs, etc. into areas which are expected to exhibit similar behavior (equivalent) so they are likely to be processed in the same way.
- Assumption: if one value works, all will work
- One from each partition better than all from one
  - Tests can be designed to cover all valid and invalid partitions. Equivalence partitioning is applicable at all levels of testing.

4.2.2 Boundary value analysis (BVA)- Phân tích giá trị biên (BVA)

- Behavior at the edge of each equivalence partition is more likely to be incorrect than behavior within the partition, so boundaries are an area where testing is likely to yield defects
- Two-point boundary: The maximum and minimum values of a partition are its boundary values
- Three boundary values: the values before, at, and just over the boundary
- Tests can be designed to cover both valid and invalid boundary values.
- Boundary value analysis can be applied at all test levels. It is relatively easy to apply and its defect- finding capability is high.
- Detailed specifications are helpful in determining the interesting boundaries.

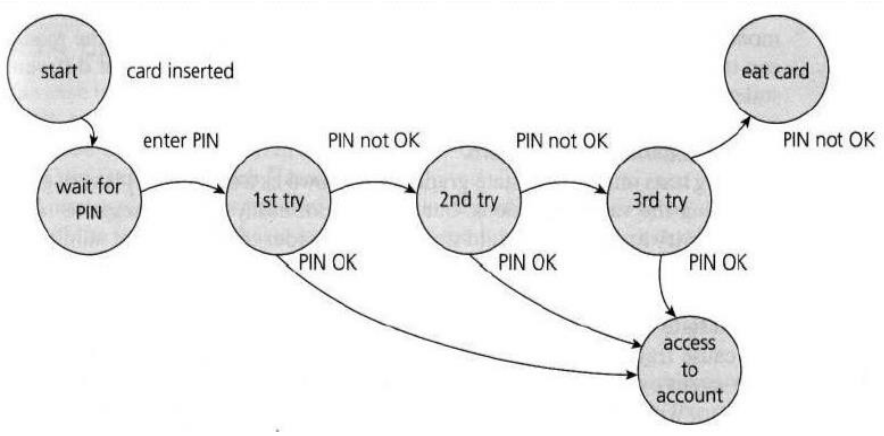


4.2.3 Decision tables - bảng quyết định

- Explore combinations of inputs, situations or events
- It is very easy to overlook specific combinations of input
- Start by expressing the input conditions of interest so that they are TRUE or FALSE

4.2.4 State transition testing - Test chuyển đổi trạng thái

- Example: State diagram for PIN entry



Switch:

0- Switch: single transition
1- Switch: couple transitions
2- Switch: triple transitions

States table – Bảng trạng thái

	Insert card	Valid PIN	Invalid PIN
S1) Start state	S2	–	–
S2) Wait for PIN	–	S6	S3
S3) 1st try invalid	–	S6	S4
S4) 2nd try invalid	–	S6	S5
S5) 3rd try invalid	–	–	S7
S6) Access account	–	?	?
S7) Eat card	S1 (for new card)	–	–

4.3 While Box test design

- The basic coverage measure is:

$$\text{Coverage} = \frac{\text{Number of coverage items exercised}}{\text{Total number of coverage items}} \times 100\%$$

4.3.1 Statement coverage

- Percentage of executable statements exercised by a test suite
  - Formula = number of statements exercised / Total number of statements

Example of statement coverage

Read (a)

If a>6 then

A=b

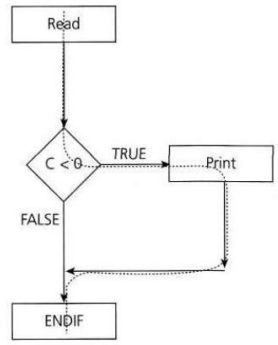
endif

We need only 1 test case a=7 to have 100% statement coverage

4.3.2 Decision coverage (Branch coverage)

- Percentage of decision outcomes exercised by a test suite

```
1 READ A
2 READ B
3 C = A - 2 * B
4 IF C <0 THEN
5       PRINT "C negative"
6 ENDIF
```

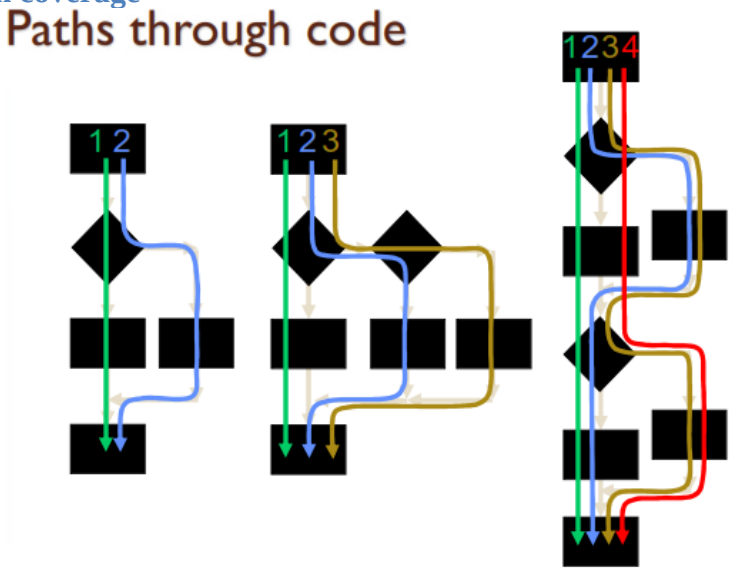


- Case 1: A=20, B-15

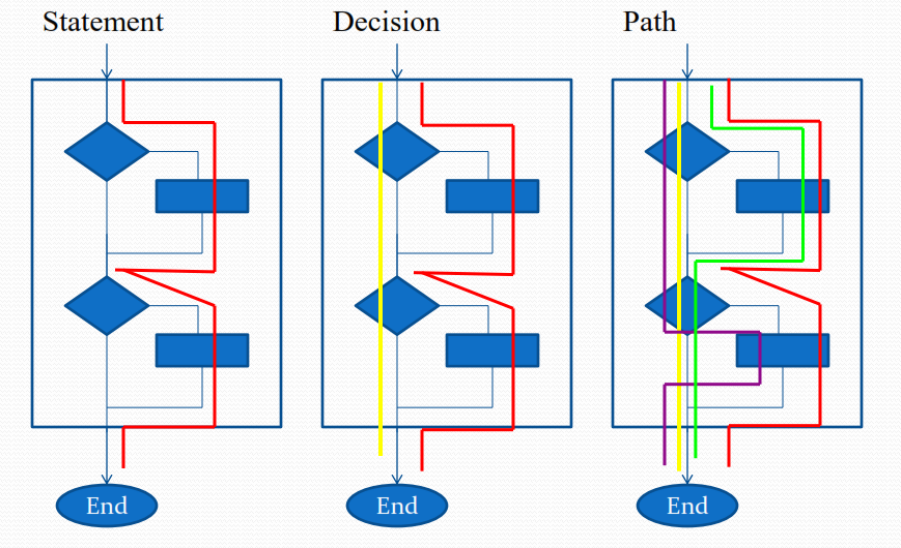
⇒False outcome isn't covered

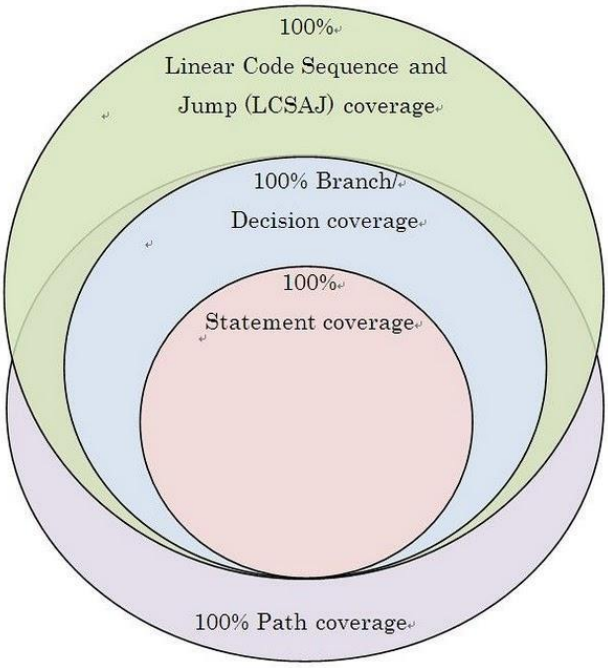
⇒Add case2: A=10, B=2

4.3.3 Path coverage



Summary while box test





4.4 Experience_base techniques (K2)

4.4.1 Error Guessing

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

- How the application has worked in the past
- What types of mistakes the developers tend to make
- Failures that have occurred in other applications
- Experience about defect and failure data
- Common knowledge about why software fails

Error guessing technique is to create a list of possible mistakes, defects, and failures, and design tests that will expose those failures and the defects that caused them

4.4.2 Exploratory Testing

In exploratory testing, informal tests are designed, executed, logged, and evaluated dynamically during test execution. The test results are used to learn more about the component or system, and to create tests for the areas that may need more testing

Exploratory testing is sometimes conducted using session-based testing to structure the activity:

- Tester uses a test charter containing test objectives within a defined time-box to guide the testing.
- Tester may use test session sheets to document the steps followed and the discoveries made.

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

4.4.3 Checklist-based Testing

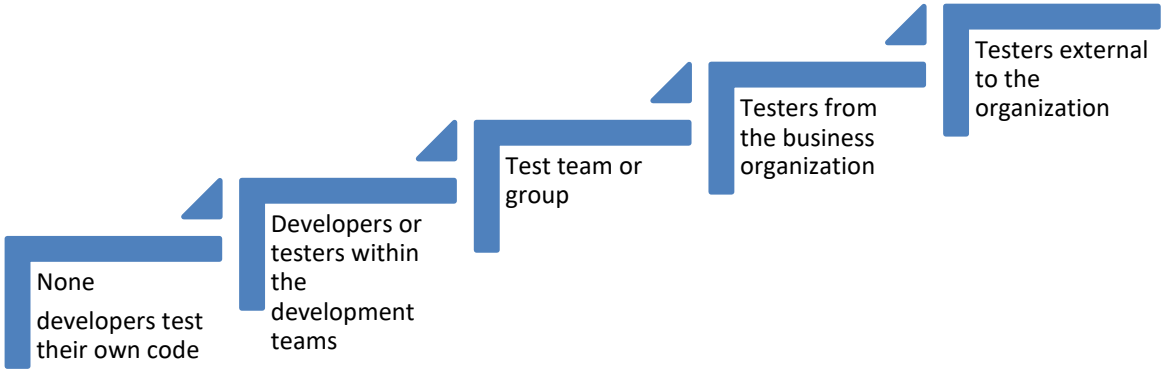
Checklists can be built based on experience, knowledge about what is important for the user, or an understanding of why and how software fails.

Checklists can be created to support various test types, including functional and non-functional testing.

In the absence of detailed test cases, checklist-based testing can provide guidelines and a degree of consistency.

Chapter 5: Test Management

5.1 Test Organization



Benefits of test independence include

- Recognize different kinds of failures
- Verify, challenge, or disprove assumptions

Drawbacks of test independence include

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

5.1.2 Tasks of a Test Manager and Tester

Testing phase	Test Manager tasks	Tester tasks
Planning	- Plan the test activities - 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	- Review and contribute to test plans - Create the detailed test execution schedule
Analysis and design Implement and execution	- Initiate the analysis, design, implementation, and execution of tests - Support the selection and implementation of tools to support the test process - Support setting up the defect management system - Set up adequate configuration management of testware	- Analyze, review, and assess requirements for testability (i.e., the test basis) - Identify and document test conditions, capture traceability - Design, set up, and verify test environment(s) - Design and implement test cases and test procedures - Prepare and acquire test data

Bản quyền thuộc về QRS-website: qr-solutions.com.vn
DC: tầng 3, số 1, ngõ 36/7 Duy Tân

	- Decide about the implementation of test environment(s)	- Execute tests, evaluate the results - Automate tests as needed - Evaluate non-functional characteristics - Review tests developed by others
Monitoring and control	- Monitor test progress and results, and check the status of exit criteria - Create test progress reports and test summary reports - Adapt planning based on test results and progress - Take any actions necessary for test control	- Use appropriate tools to facilitate the test process
Organization task	- Develop or review a test policy and test strategy - Introduce suitable metrics for measuring test progress and evaluating the quality of the testing and the product - Promote and advocate the testers, the test team - Develop the skills and careers of testers	-

5.2 Test Planning and Estimation

5.2.1 Purpose and Content of 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
- Scheduling of test analysis, design, implementation, execution, and evaluation activities
- Selecting metrics for test monitoring and control
- Budgeting for the test activities
- Determining the level of detail and structure for test documentation

5.2.2 Test Strategy and Test Approach

Analytical: 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. Examples of such models include business process models, state models, and reliability growth models

Methodical: 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 standards for mobile apps or web pages

Process-compliant (or standard-compliant): based on external rules and standards, process

Directed (or consultative): base on the advice, guidance, or instructions of stakeholders, business domain experts, or technology experts

Regression-averse: highly automation of regression tests, and standard test suites

Reactive: Tests are designed and implemented, and may immediately be executed in response to knowledge gained from prior test results. Exploratory testing is a common technique employed in reactive strategies

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

Typical entry criteria include:

- 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

Typical exit criteria include:

- Planned tests have been executed
- A defined level of coverage
- 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

5.2.4 Test Execution Schedule - Lịch trình thực hiện

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.

5.2.5 Factors Influencing the Test Effort

Test effort estimation involves predicting the amount of test-related work that will be needed in order to meet the objectives of the testing for a particular project, release, or iteration

Product characteristics

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

Development process characteristics

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

People characteristics

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

Test results

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

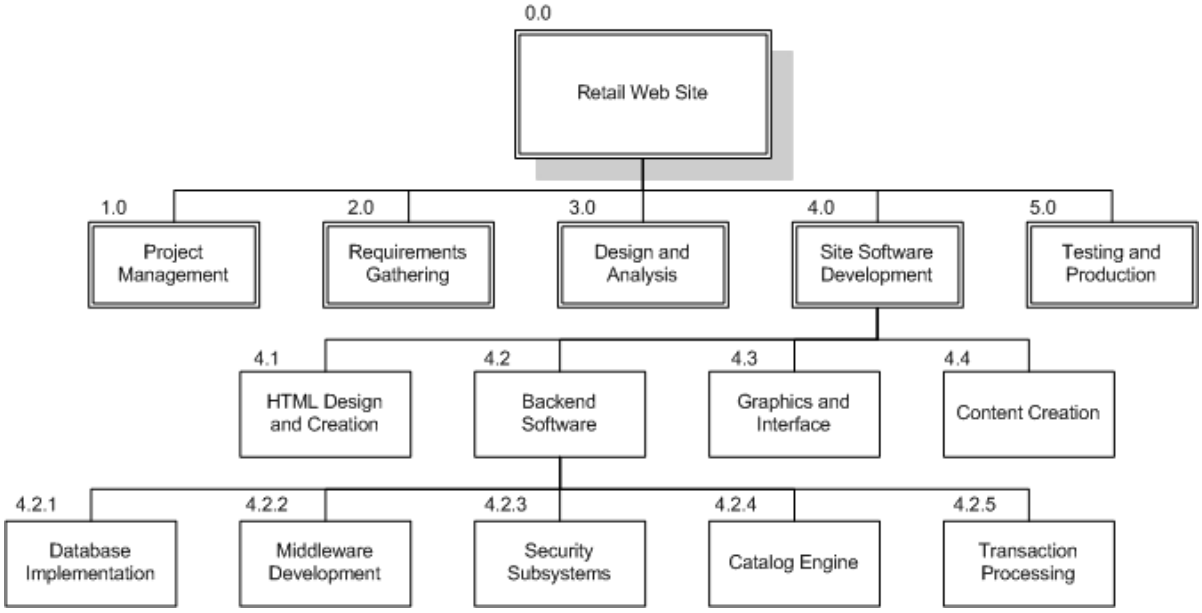
Two of the most commonly used techniques are:

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

	Project management	Requirement development	Design	Coding	Testing	Quality assurance	CM	Risk buffer
Average 100 old projects	10%	15%	10%	25%	35%	2.5%	2.5%	10%
Project A	?	?	?	1250pd	??	?	?	

2. The expert-based technique: estimating the test effort based on the experience of the owners of the testing tasks or by experts

WBS- Work Breakdown Structure



5.3 Test Monitoring and Control

Test monitoring is to gather information and provide feedback and visibility about test activities, measure and assess test progress and measure whether the test exit criteria

Test control describes any guiding or corrective actions taken:

- Re-prioritizing tests when an identified risk occurs (e.g., software delivered late)
- Changing the test schedule due to availability or unavailability of a test environment or other resources
- Re-evaluating whether a test item meets an entry or exit criterion due to rework

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

Common test metrics include:

-
- Percentage of planned work done in each test activities
- Test case execution (e.g., number of test cases run/not run, test cases passed/failed, and/or test conditions passed/failed)
-
- Defect information (e.g., defect density, defects found and fixed, failure rate, and confirmation test results)
-
- Test coverage of requirements, user stories, acceptance criteria, risks, or code
- Task completion, resource allocation and usage, and effort
-
- Cost of testing, including the cost compared to the benefit of finding the next defect or the cost compared to the benefit of running the next test

5.3.2 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

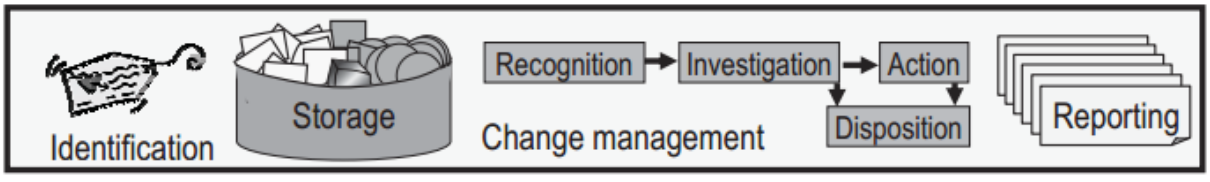
Content common to test progress reports:

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

Typical test progress reports and test summary reports may include:

- Summary of testing performed
- Analysis the information on what occurred during a test period
- Deviations from plan
-
- Metrics of defects, test cases, test coverage, activity progress, and resource consumption.
- Residual risks
- Reusable test work products produced

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:

- 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

Bản quyền thuộc về QRS-website: qr-solutions.com.vn
DC: tầng 3, số 1, ngõ 36/7 Duy Tân

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

5.5 Risks and Testing

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

Project risk	Product risk (quality risks)
A risk to the project's capability to deliver products: Scope, cost, Time	A risk to quality of product
Related to management and control of the (test) project: - Skill, training and staff shortages - Personnel issues - Political issues - Technical issues - Schedules - Budget	Directly related to the test object - Failure-prone software delivered - The potential that the software/hardware could cause harm to an individual or company - Poor software characteristics (e.g., functionality, reliability, usability and performance) - Poor data integrity and quality - Software that does not perform its intended functions - User experience (UX) feedback might not meet product expectations - A loop control structure may be coded incorrectly - A particular computation may be performed incorrectly in some circumstances

5.5.3 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.

Testing is used to reduce the probability of an adverse event occurring, or to reduce the impact of an adverse event.

Testing is used as a risk mitigation activity, to provide feedback about identified risks, as well as providing feedback on residual (unresolved) risks

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

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

Risk activities include:

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

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

5.6 Defect Management

Since one of the objectives of testing is to find defects, the discrepancies between actual and expected outcomes need to be logged as incidents. An incident must be investigated and may turn out to be a defect.

Incident reports have the following objectives:

- Provide developers and other parties with feedback about the problem to enable identification, isolation and correction as necessary
- Provide test leaders a means of tracking the quality of the system under test and the progress of the testing
- Provide ideas for test process improvement

A defect report filed during dynamic testing typically includes:

- An identifier
- A title and a short summary of the defect being reported
- Date of the defect report, issuing organization, and author
- Identification of the test item (configuration item being tested) and environment
- The development lifecycle phase(s) in which the defect was observed
- A description of the defect to enable reproduction and resolution, including logs, database dumps screenshots, or recordings
-
- Expected and actual results
- Scope or degree of impact (severity) of the defect on the interests of stakeholder(s) Urgency/priority to fix
- State of the defect report (e.g., open, deferred, duplicate, waiting to be fixed, awaiting confirmation testing, re-opened, closed)
-
- Conclusions, recommendations and approvals
- Global issues, such as other areas that may be affected by a change resulting from the defect
- Change history, such as the sequence of actions taken by project team members with respect to the defect to isolate, repair, and confirm it as fixed
-
- References, including the test case that revealed the problem

Refer: ISO standard (ISO/IEC/IEEE 29119-3)

Chapter 6: Tool Support for Testing

6.1 Test Tool Considerations

6.1.1 Test Tool Classification

Test tools can have some purposes:

- Improve the efficiency of test activities by automating repetitive tasks or tasks that require significant resources when done manually (e.g., test execution, regression testing)
- Improve the efficiency of test activities by supporting manual test activities throughout the test process
- Improve the quality of test activities by allowing for more consistent testing and a higher level of defect reproducibility
- Automate activities that cannot be executed manually (e.g., large scale performance testing)
- Increase reliability of testing (e.g., by automating large data comparisons or simulating behavior)

Tool categories

Support management
- Test management tools
- Requirements management tools
- Defect management tools
- Configuration management tools
- Continuous integration tools (D)

Support for static testing
- Tools that support reviews
- Static analysis tools (D)

Support for test execution and logging
- Test execution tools
- Coverage tools
- Test harnesses (D)
- Unit test framework tools (D)

Support for performance measurement and dynamic analysis
- Performance testing tools
- Monitoring tools
- Dynamic analysis tools (D)

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

Support for specialized testing needs
- Data quality assessment
- Data conversion and migration
- Usability testing
- Accessibility testing
- Localization testing
- Security testing
- Portability testing

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
- Greater consistency and repeatability (e.g., test data is created in a coherent manner, tests are executed by a tool in the same order with the same frequency, and tests are consistently derived from requirements)
- More objective assessment (e.g., static measures, coverage)
- Easier access to information about testing (e.g., statistics and graphs about test progress, defect rates and performance)

Potential risks of using tools to support testing include:

- Expectations for the tool may be unrealistic (including functionality and ease of use)
- The time, cost and effort for the initial introduction of a tool may be under-estimated (including training and external expertise)
- The time and effort needed to achieve significant and continuing benefits from the tool may be under-estimated (including the need for changes in the test process and continuous improvement in the way the tool is used)
- The effort required to maintain the test assets generated by the tool may be under-estimated
- The tool may be relied on too much (seen as a replacement for test design or execution, or the use of automated testing where manual testing would be better)
- Version control of test assets may be neglected
- Relationships and interoperability issues between critical tools may be neglected, such as requirements management tools, configuration management tools, defect management tools and tools from multiple vendors
- The tool vendor may go out of business, retire the tool, or sell the tool to a different vendor
- The vendor may provide a poor response for support, upgrades, and defect fixes
- An open source project may be suspended
- A new platform or technology may not be supported by the tool
- There may be no clear ownership of the tool (e.g., for mentoring, updates, etc.)

6.1.3 Special Considerations for Test Execution and Test Management Tools

Test execution tools

Data-driven scripting technique	Keyword-driven scripting technique	Model-Based testing (MBT)
Data files store test input and expected results in table or spreadsheet	data files store test input, expected results and keywords in table or spreadsheet	a functional specification to be captured in the form of a model, such as an activity diagram
support capture/playback tools	writing script manually	The MBT tool interprets the model in order to create test case specifications which can then be saved in a test management tool and/or executed by a test execution tool

Example keyword driven:

Keyword	User	Password	Result
Add_User	User1	Pass1	User added message
Add_User	@Rec34	@Rec35	User added message
Reset_Password	User1	Welcome	Password reset confirmation message
Delete_User	User1		Invalid username/password message
Add_User	User3	Pass3	User added message
Delete_User	User2		User not found message

Test management tools

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

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

6.2 Effective Use of Tools

6.2.1 Main Principles for Tool Selection

- Assessment of organizational maturity, strengths and weaknesses and identification of opportunities for an improved test process supported by tools
- Understanding of the technologies used by the test object(s), in order to select a tool that is compatible with that technology
- The build and continuous integration tools already in use within the organization, in order to ensure tool compatibility and integration
- Evaluation of the tool against clear requirements and objective criteria
- Consideration of whether or not the tool is available for a free trial period (and for how long)
- Evaluation of the vendor (including training, support and commercial aspects) or support for non-commercial (e.g., open source) tools
- Identification of internal requirements for coaching and mentoring in the use of the tool
- Evaluation of training needs, considering the testing (and test automation) skills of those who will be working directly with the tool(s)
- Consideration of pros and cons of various licensing models (e.g., commercial or open source)
- Estimation of a cost-benefit ratio based on a concrete business case (if required)

6.2.2 Pilot Projects for Introducing a Tool into an Organization

- Gaining in-depth knowledge about the tool, understanding both its strengths and weaknesses
- Evaluate how the tool fits with existing processes and practices, and determine what would need to change
- Decide on standard ways of using, managing, storing and maintaining the tool and the test assets
- Assess 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

- 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 and coaching/mentoring for new users
- Defining usage guidelines
- Implementing a way to gather usage information from the actual use
- Monitoring tool use and benefits

- Providing support for the test team for a given tool
- Gathering lessons learned from all teams