ISTQB Foundation Level Summary

Chapter 1: Fundamentals of Testing

1.1 What is Testing?

- **Software Testing is a way to:**
  - Assess the quality of the software.
  - Reduce the risk of software failure in operation.
- **Software Testing ≠ Test Execution.** Software testing is a process which includes many different activities.
- Testing not only focuses on verification of **requirements**, **user stories**, or **other specifications**. It also involves validation which is checking whether the **system will meet user and other stakeholder needs**.

1.1.1 Typical Objectives of Testing

- Work products evaluation.
- Requirements fulfillment.
- Building confidence.
- Prevent defects.
- Find failures and defects.
- Providing information to stakeholders.
- Reduce the level of risk.
- Compliance with laws and regulations.

The objectives of testing can vary, depending upon the context of the component or system being tested, the test level, and the software development lifecycle model.

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**.
- Confirmation testing checks whether the fixes resolved the defects.
- In Agile development and in some other lifecycles, testers may be involved in debugging and component testing.

1.2.1 Testing's Contributions to Success

- Having testers involved in **requirements reviews or user story**. The identification and removal of requirements defects reduces the risk of incorrect or untestable functionality being developed.
- Having testers work closely with **system designer**. This increased understanding can reduce the risk of fundamental design defects and enable tests to be identified at an early stage.
- Having testers work closely with **developers while the code is under development**. 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**. This increases the likelihood that the software meets stakeholder needs and satisfies requirements.

1.2.2 Quality Assurance and Testing

- Quality management includes activities that control an organization quality (quality assurance and quality control).
- Quality assurance is typically focused on adherence to proper processes.
  - Proper processes -> Work products higher quality -> Defect Prevention
- Quality control involves various activities, including test activities, that support the achievement of appropriate levels of quality.

1.2.3 Errors, Defects, and Failures

- Error (**mistake**) -> Defect (**Fault or Bug**) -> Error -> Defect in a related work product.
- Defect in the code is executed -> Failure.
- **Errors may occur for many reasons, such as:**
    - Time pressure.
    - Human fallibility.
    - Lack of experience.
    - Miscommunication.
    - Complexity of work products.
    - Misunderstandings system interactions.
    - New technologies.
    - environmental conditions.
- **False Positive**: Occur due to Errors in the way tests were executed, or due to defects in the test data, the test environment, or other test-ware, or for other reasons. reported as defects but aren't actually defects.
- **False Negative**: Tests that do not detect defects that they should have detected.

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.
- Root cause analysis can lead to process improvements that prevent a significant number of future defects from being introduced.

1.3 Seven Testing Principles

1- **Testing shows the presence of defects, not their absence**: Testing can show that defects are present but cannot prove that there are no defects.

2- **Exhaustive testing is impossible:** Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases.

3- **Early testing saves time and money (Shift Left):** To find defects early, both static and dynamic test activities should be started as early as possible.

4- **Defects Cluster Together:** A small number of modules usually contains most of the defects discovered during pre-release testing.

5- **Beware of Pesticide Paradox:** If the same tests are repeated over and over again, eventually these tests no longer find any new defects.

6- **Testing is Context Dependent:** Testing is done differently in different contexts (e.g. Testing in **Agile** and **Sequential** lifecycle project).

7- **Absence-of-errors is a fallacy:** It is a fallacy to expect that just finding and fixing many defects will ensure the success of a system.

1.4.1 Test Process in Context

- **Factors that influence the test process:**
    - Software development lifecycle model.
    - Test levels and test types.
    - Product and project risks.
    - Business domain.
    - Operational constraints (**Budgets and resources** - **Timescales** - **Complexity** - **Contractual and regulatory**).
    - Organizational policies and practices.
    - Required internal and external standards.
- It is very useful if the test basis has measurable coverage criteria defined.

- The coverage criteria can act effectively as key performance indicators (**KPIs**) to drive the activities that demonstrate achievement of software test objectives.
- The **coverage criteria** may require at least one test case for each element of the test basis.

1.4.2 Test Activities and Tasks

Many test activity groups may appear logically sequential, they are often implemented iteratively (e.g., **Agile Development**).

1- **Test Planning:**
   - Defining Objectives of Testing & the approaches to meet these objectives.
   - Test Plans may be revisited.

2- **Test Monitoring & Control:**
   - On-going comparison of actual progress against the test plan using test monitoring metrics in the test plan.
   - Taking actions necessary to meet the objectives of the test plan.
   - Supported by evaluation of exit criteria or DOD (Definition of done) by:
     - 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**.

3- **Test Analysis:**
   - Answer the question "What to test?".
   - Analyze the Test Basis (**Requirement Specifications** – **Design** – **Code** – **Database** – **Risk Analysis reports**).
   - Define & prioritize Test Conditions.
   - Capturing bi-directional traceability (between test basis and the associated test conditions).
   - Test conditions can be used as objectives in Test Charters.

4- **Test Design:**
   - Answer the question "How to Test?".
   - Design and prioritize Test Cases.
   - Identify Test Data.
   - Design Test Environment.

5- **Test Implementation:**
   - Answer the question "Do we now have everything in place to run the tests?".
   - Develop and prioritize Test Procedures.
   - Create Test suites, arrange them in Test Execution Schedule.
   - Build Test Environment.
   - Preparing test data and ensuring it is properly loaded in the test environment.
   - Test design and test implementation tasks are often combined.

6- **Test Execution:**
   - Run the Test Suites.
   - Compare actual result with expected result.
   - Report Defects.
   - Confirmation & Regression Testing.

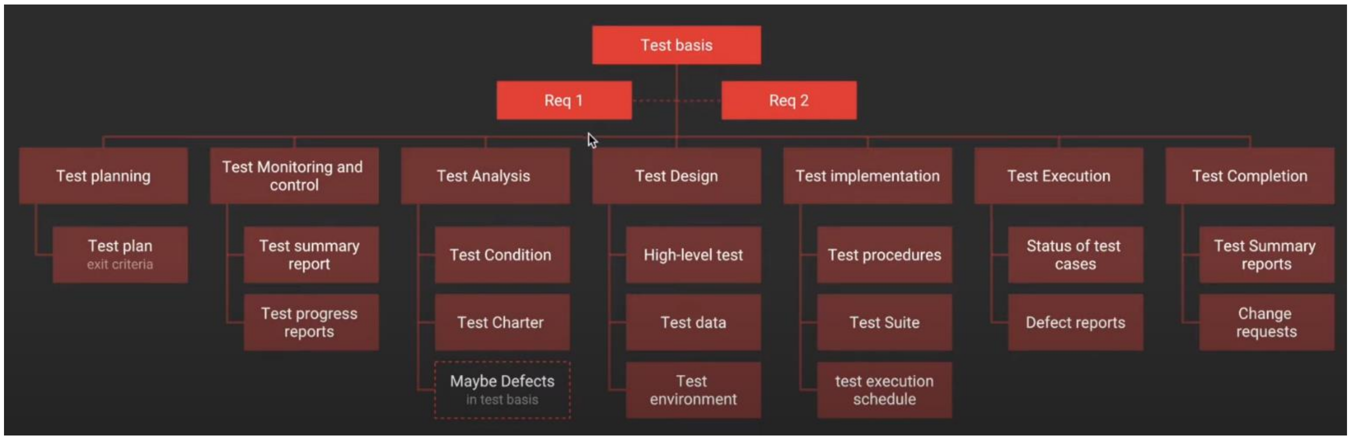
7- **Test Completion:**
   - Collect data from completed test activities.
   - Occurs at project milestones.
   - Check Defect Reports.
   - Create Test Summary Report.
   - Hand over the test-ware.

1.4.3 Test Work Products

Activity	Work Products
Test Planning	Test Plans
Test Monitoring & Control	Test Progress/Summary Reports
Test Analysis	Test Conditions – Test Charters
Test Design	Test Cases – Test Data
Test Implementation	Test Procedures – Test Suites – Test Execution Schedule
Test Execution	Status of Test Cases – Defect Reports
Test Completion	Test Summary Report – Change Requests

1.4.4 Traceability between the Test Basis and Test Work Products

- Analyzing the impact of changes.
- Making testing auditable.
- Meeting IT governance criteria.
- Improving the understandability of test progress reports and test summary reports (e.g., requirements that passed their tests, requirements that failed their tests, and requirements that have pending tests)



1.5.1 Human Psychology and Testing

Testers and test managers need to have good interpersonal skills to be able to communicate effectively:

- Start with collaboration rather than battles.
- 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. Write objective and factual defect reports and review findings.
- Try to understand how the other person feels and why 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

A tester's mindset should include **curiosity**, **professional pessimism**, a **critical eye**, **attention to detail**, and **motivation** for **good and positive communications and relationships**.

Chapter 2: Testing Throughout the Software Development Lifecycle

2.1.1 Software Development and Software Testing

- **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.
- **Sequential Development Model:**
  - Describes the software development process as a linear, sequential flow of activities.
  - Any phase in the development process should begin when the previous phase is complete.
  - In theory, there is no overlap of phases, but in practice, it is beneficial to have early feedback from the following phase.
  - **Waterfall Model:**
    - The development activities are completed one after another.
    - Test activities only occur after all other development activities have been completed.
  - **V-Model:**
    - Unlike the Waterfall model, the V-model integrates the test process throughout the development process, implementing the principle of early testing.
- **Incremental Development Model:**
  - Involves establishing requirements, designing, building, and testing a system in pieces, which means that the software's features grow incrementally.
  - The size of these feature increments varies, with some methods having larger pieces and some smaller pieces.
- **Iterative Development Model:**
  - Occurs when groups of features are specified, designed, built, and tested together in a series of cycles, often of a fixed duration.
  - Iterations may involve changes to features developed in earlier iterations, along with changes in project scope.
  - Each iteration delivers working software which is a growing subset of the overall set of features until the final software is delivered or development is stopped.
    - **Rational Unified Process**: Each iteration tends to be relatively long (e.g., two to three months), and the feature increments are correspondingly large, such as two or three groups of related features.
    - **Scrum**: Each iteration tends to be relatively short (e.g., hours, days, or a few weeks), and the feature increments are correspondingly small, such as a few enhancements and/or two or three new features.
    - **Kanban**: Implemented with or without fixed-length iterations, which can deliver either a single enhancement or feature upon completion or can group features together to release at once.
    - **Spiral (Prototyping)**: Involves creating experimental increments, some of which may be heavily re-worked or even abandoned in subsequent development work.
- **Continuous Delivery:**
  - In some cases, teams use continuous delivery or continuous deployment, both of which involve significant automation of multiple test levels as part of their delivery pipelines.
  - Many development efforts using these methods include the concept of self-organizing teams.
  - Regression testing is increasingly important as the system grows.

Note: Software development lifecycle models themselves may be combined.

2.2 Test Levels

Point	Component Testing	Integration Testing	System Testing	Acceptance Testing
Focuses On	components that are separately testable	interactions between components or systems	The behavior and capabilities of a whole system or product – The end-to-end tasks and the non-functional behaviors	The behavior and capabilities of a whole system or product
Objectives	Reduce Risk – Find Defects – Prevent Defects Escape to Higher Test Levels – Verify Behavior – Build Confidence			Establish confidence – Validate System – Verify behavior
Test Basis	Detailed design – Code – Data Model – Component Specification	Software & System Design – Sequence Diagrams – Use Cases – Workflows	SRS – Risk Analysis Reports – Use Cases – Epics – User Stories – State Diagrams - Manuals	Business processes – Requirements – Use Cases – Installation Procedures – Risk Analysis Reports – Regulations – Contracts
Test Objects	Components – Units – Modules – Code – Data Structures – Classes	Subsystems – APIs – Microservices – Interfaces – Databases – Infrastructure	Applications – Operating Systems Under Test – System Configuration	System Under Test – Recovery Systems – Hot Sites – Forms – Reports
Defects & Failures	Incorrect functionality – Data flow problems – Incorrect code & Logic	Failures in communication – Incorrect data – Interface mismatch – Incorrect Data	System failure – Incorrect data flow – Unexpected behavior	System workflow – Business rules – Contract – Non-functional failures
Approaches & Responsibilities	Done by the developer – TDD	Top down (**Stubs**) – Bottom up (**Drivers**)		

2.2.1 Component Testing

- Also known as unit or module testing.
- **Test Driven Development (TDD):** Developing automated test cases, then building and integrating small pieces of code, then executing the component tests, correcting any issues, and re-factoring the code.

2.2.2 Integration Testing

- **Component integration tests:**
  - Should focus on communication between the modules, not the functionality of the individual modules, as that should have been covered during component testing.
  - Performed after component testing and is generally automated.
  - Often the responsibility of Developers.
- **System integration tests:**
  - Should focus on communication between the systems, not the functionality of the individual systems, as that should have been covered during system testing.
  - May be done after system testing or in parallel.
  - Often the responsibility of Testers.
- Integration should normally be incremental (i.e., a small number of additional components or systems at a time) rather than "big bang" (i.e., integrating all components or systems in one single step).
- In the **Top-Down** approach, the top module is ready, and in the bottom module not yet (using stubs instead), and in the **Bottom-Up** approach bottom module is ready, and in the top module not yet (using drivers instead).

2.2.3 System Testing

- Produces information that is used by stakeholders to make release decisions.
- The test environment should ideally correspond to the final target or production environment.
- Independent testers typically carry out system testing.

2.2.4 Acceptance Testing

- May produce information to assess the system's readiness for deployment and use by the customer (**end-user**).
- Defects may be found during acceptance testing, but finding defects is often not an objective, and **finding a significant number of defects during acceptance testing** may in some cases be considered a **major project risk**.
- Often done by customers, business users, product owners, or operators of a system, and other stakeholders.
- **Types of Acceptance Test:**
  - **User Acceptance Test (UAT):** building confidence that the users can use the system to meet their needs, fulfill requirements, and perform business processes with minimum difficulty, cost, and risk.
  - **Operational Acceptance Test (OAT):** building confidence that the operators or system administrators can keep the system working properly for the users in the operational environment.
  - **Contractual and regulatory acceptance testing:** Building confidence that contractual or regulatory compliance has been achieved.
  - **Alpha and Beta Testing:** Building confidence among potential or existing customers that they can use the system under normal, everyday conditions.
    - **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.

2.3 Test Types

2.3.1 Functional Testing

- Involves tests that evaluate functions that the system should perform.
- The functions are "what" the system should do?
- Considers the behavior of the software, so black-box techniques may be used to derive test conditions and test cases for the functionality of the component or system.
- The thoroughness of functional testing can be measured through functional coverage.

2.3.2 Non-functional Testing

- Evaluates characteristics of systems and software such as **usability**, **performance**, **efficiency,** or **security**.
- Is the testing of "how well" the system behaves?
- Should be done as early as possible.
- The thoroughness of non-functional testing can be measured through non-functional coverage.

2.3.3 White-box Testing

- Based on the system's internal structure (**code**, **architecture**, **workflows**, and/or **data flows**) or implementation.
- The thoroughness of white-box testing can be measured through structural coverage.

2.3.4 Change-related Testing

- **Confirmation Testing (Retesting):**
  - After a defect is fixed, the software should be tested with all test cases that failed due to the defect and may also be tested with new tests.
  - The purpose of a confirmation test is to confirm whether the original defect has been successfully fixed.
- **Regression Testing:**
  - When a change (**fix** or **another type of change**) is made it may accidentally affect the behavior of other parts of the code within the same component, in other components of the same system, or even in other systems.
  - Regression test suites are run many times and generally evolve slowly, so regression testing is a strong candidate for automation.

Note: It is possible to perform any of the test types mentioned above at any test level.

2.4 Maintenance Testing

- Once deployed to production environments, software and systems need to be maintained (**Fix defects discovered in operational use**, to **add new functionality**, **delete or alter already-delivered functionality,** or to **improve non-functional quality characteristics)**.
- 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**).
- **Scope of maintenance testing depends on:**
  - The degree of risk of the change.
  - The size of the existing system.
  - The size of the change.

2.4.1 Triggers for Maintenance

- **Modification**, such as planned enhancements, corrective changes, changes of the operational environment.
- **Migration**, such as from one platform to another, 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.
- The side effects and affected areas in the system need to be tested for regressions.
- Impact analysis may be done before a change is made, to help decide if the change should be made.

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). without 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, for example:

- Specifications, requirements.
- Epics, user stories, and acceptance criteria.
- Architecture and design specifications.
- Code.
- Test-ware, including test plans, test cases, test procedures, and automated test scripts.
- User guides.
- Web pages.
- Contracts, project plans, schedules, and budgets.
- Models, such as activity diagrams, which may be used for Model-Based testing.

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.
- Increasing development productivity (e.g., due to improved design, more maintainable code).
- Reducing development and testing cost and time.
- Reducing total cost of quality over the software's lifetime, due to fewer failures after delivery into operation.
- Improving communication between team members while participating in reviews.

3.2.1 Work Product Review Process

The review process comprises the following main activities:

- **Planning**:
  - Defining the scope (**purpose**, **documents**, **quality characteristics to be evaluated**).
  - Estimating effort and timeframe.
  - Identifying review characteristics (**types**, **roles**, **activities**, and **checklists**)
  - Selecting the people to participate in the review and allocating roles.
  - Defining the entry and exit criteria.
  - Checking that entry criteria are met.
- **Initiate Review**:
  - Distributing the work product.
  - 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.
- **Fixing and Reporting**:
  - Creating defect reports.
  - Fixing defects found (**author**).
  - Communicate defects to the appropriate person or team.
  - Recording updated status of defects.
  - Gathering metrices.
  - Checking that exit criteria are met.
  - Accepting the work product when the exit criteria are reached.

3.2.2 Roles and responsibilities in a formal review

A typical formal review will include the roles below:

- **Author**:
  - Creates the work product under review.
  - Fixes defects in the work product (**if necessary**).
- **Management**:
  - Responsible for review planning.
  - Executes control decisions in the event of inadequate outcomes.
  - Decides on the execution of reviews.
  - Assigns **staff**, **budget**, and **time**.
  - Monitors ongoing cost-effectiveness.
- **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.
- **Scribe (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

POC	Informal (e.g., Pairing)	Walkthrough	Technical	Inspection
Individual Preparation	N	O	Y	Y
Review Meeting	O	Y	O	Y
Checklist	O	O	O	Y
Scriber	N	Y	Y	Y
Review Logs	N	O	Y	Y
Led by	College of the author	Author	Facilitator	Facilitator
Additional	Common in Agile	Take form of (**Scenarios**, **Dry Runs** or **Simulations**)		May include a **dedicated reader**
		From quite Informal to very formal		Root cause analysis - Metrices

3.2.4 Applying Review Techniques

Technique	Explanation
Ad hoc	Reviewers with little or no guidance read through the work and report issues as they find them. It is commonly used and requires little preparation but may lead to duplicate issue
Checklist-Based	A semantic technique, whereby reviewers follow a checklist of potential defects, that is distributed at review initiation (e.g., by the facilitator)
Scenarios and Dry Runs	Reviewers with structured guidelines on how to read through the work product with better guidelines on how to identify specific defect types than simple checklist.
Role-Based	Reviewers assess the work from the perspective of different stakeholders, such as end users (**senior**, **child**) or specific roles in the organization (**user administrator**, **performance tester**)
Perspective-Based	Reviewers examine the work from various viewpoints like **end users**, **marketers**, **designers**, **testers**, or **operations personnel.** perspective-based reading is the most effective technique.

Note: Reviewers should not be constrained to checklists or documented scenarios.

3.2.5 Success Factors for Reviews

Organizational Success Factors	People-related Success Factors
1- Each review has clear objectives.	1- The right people are involved.
2- Appropriate review types to work product.	2- Testers are seen as valued reviewers.
3- Suitable review techniques for defects identification.	3- Participants dedicate time and attention to detail.
4- All checklists used are up to date.	4- Reviews are conducted on small chunks.
5- Large documents are in small chunks.	5- Defects found are acknowledged, appreciated.
6- Participants have adequate time to prepare.	6- The meeting is well-managed.
7- Reviews are scheduled with adequate notice.	7- The review is in an atmosphere of trust.
8- Management supports the review process.	8- Participants avoid body language.
	9- Adequate training is provided.
	10- A culture of learning and process improvement.

Chapter 4: Test Techniques

4.1.2 Categories of Test Techniques and Their Characteristics

Technique	Test Basis	
Black-Box (Behavioral – Behavioral-based)	Requirements - Use Cases - User Stories	Concrete on the inputs and outputs of the test object without reference to its internal structure
White-Box (Structural – Structural-based)	Architecture analysis - Detailed Design - Internal Structure - Code	Concrete on the structure and processing within the test object
Experience-Based	Knowledge of (Testers - Developers - Users)	Often combined with black-box and white-box test techniques

4.2 Black-box Test Techniques

4.2.1 Equivalence Partitioning (EP)

- Partitions are called equivalence classes.
- Partition that contains valid values is called a “valid equivalence partition”.
- Partition that contains invalid values is called an “invalid equivalence partition”.
- Each value must belong to one and only one equivalence partition.
- Any partition may be divided into sub partitions if required.
- Coverage is measured as the number of equivalence partitions tested by at least one value, divided by the total number of identified equivalence partitions.
- When invalid equivalence partitions are used, they should be tested individually.

4.2.2 Boundary Value Analysis (BVA) “Default Two Point Value if not asking for three”

- An extension of (EP) but can only be used when the partition is ordered, consisting of numeric or sequential data.
- The minimum and maximum values (**first** and **last** values) of a partition are its boundary values.
- Behavior at the boundaries of equivalence partitions is likely to be incorrect than behavior within the partitions.
- Some variations identify three boundary values per boundary: the value before, at, and just over the boundary.
- This technique is generally used to test requirements that call for a range of numbers (including **dates** and **times**).
- Boundary coverage for a partition is measured as the number of boundary values tested, divided by the total number of identified boundary test values.

4.2.3 Decision Table Testing

- Decision tables are a good way to record complex business rules that a system must implement.
- Conditions (often **inputs**) and the resulting actions (often **outputs**) of the system. These form the rows of the table, usually with the conditions at the top and the actions at the bottom.
- Each column corresponds to a decision rule that defines a unique combination of conditions.
- The values of the conditions and actions are usually shown as Boolean values (**true** or **false**) or discrete values (e.g., **red**, **green**, **blue**), but can also be numbers or ranges of numbers.
- The table can be collapsed by deleting columns containing impossible combinations of conditions, columns containing possible but infeasible combinations of conditions, and columns that test combinations of conditions that do not affect the outcome.
- The common minimum coverage standard for is to have at least one test case per decision rule in the table.
- Coverage is measured as the number of decision rules tested, divided by the total number of decision rules.
- **Trick:** Minimum Number of Test cases = Number of actions + 1.

4.2.4 State Transition Testing

- A **state transition diagram** shows the possible software states, as well as **how the software** enters, exits, and transitions between states.
- A transition is initiated by an event (e.g., **user input of a value into a field**).
- The state change may result in the software taking an action (e.g., **outputting a calculation or error message**)
- A **state transition table** shows all valid transitions and potentially invalid transitions between states.
- **State transition diagrams** normally show only the valid transitions and exclude the invalid transitions.
- Coverage is commonly measured as the number of identified states or transitions tested, divided by the total number of identified states or transitions in the test object.

4.2.5 Use Case Testing

- Use cases are associated with actors (human users, external hardware, or other components or systems) and subjects (the component or system to which the use case is applied).
- Each use case specifies some behavior that a subject can perform in collaboration with one or more actors.
- A use case can be described by interactions and activities, as well as preconditions, postconditions and natural language where appropriate.
- A use case can include possible variations of its basic behavior, including exceptional behavior and error handling (system response, and communication errors, e.g., resulting in an error message).
- Tests are designed to exercise defined behaviors (basic, exceptional, or alternative, and error handling).
- Coverage can be measured by the use case behaviors tested divided by the total number of use case behaviors.

4.3 White-box Test Techniques

4.3.1 Statement Testing and Coverage

- Statement testing exercises the executable statements in the code.
- Coverage is measured as the number of statements executed by the tests divided by the total number of executable statements.

4.3.2 Decision Testing and Coverage

- Decision testing exercises the decisions in the code that are executed based on the decision outcomes.
- For an IF statement, **one** for the **true** outcome and **one** for the **false** outcome; for a CASE statement, test cases would be required for all the possible outcomes, including the default outcome).
- Coverage is measured as the number of decision outcomes executed by the tests divided by the total number of decision outcomes in the test object.

Note: Achieving 100% decision coverage guarantees 100% statement coverage (but **not vice versa**).

4.4 Experience-based Test Techniques

- When applying experience-based test techniques, the test cases are derived from the tester's skill and intuition, and their experience with similar applications and technologies.
- These techniques can be helpful in identifying tests that were not easily identified by other more systematic techniques.
- Coverage can be difficult to assess and may not be measurable with these techniques.

4.4.1 Error Guessing

- **A technique used to anticipate the occurrence of mistakes, defects, and failures, based on the tester's knowledge, including:**
    - How has the application worked in the past?
    - What types of mistakes the developers tend to make?
    - Failures that have occurred in other applications.
- Create a list of **possible mistakes**, **defects**, and **failures**, and design tests that will expose those failures and the defects that caused them.
- These **mistakes, defects, failure** lists can be **built based on** experience, defect, and failure data, or from common knowledge about why software fails.

4.4.2 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.
- Sometimes conducted using session-based testing, where test is conducted within a defined time-box, and the tester uses a test charter containing test objectives to guide the testing.
- Most useful when there are few or inadequate specifications or significant time pressure on testing.

4.4.3 Checklist-based Testing

- Testers **design**, **implement**, and **execute** tests to cover test conditions found in a checklist.
- As part of analysis, testers create a new checklist or expand an existing checklist, but testers may also use an existing checklist without modification.
- Such checklists can be built based on experience, knowledge about what is important for the user, or an understanding of why and how software fails.
- Checklists can be created to support various test types, including functional and non-functional testing.
- In the absence of detailed test cases, checklist-based testing can provide guidelines and a degree of consistency.

Chapter 5: Test Management

5.1.1 Independent Testing

- **Testing's Degree of Independence:**
  1- No independent testers; developers testing their own code.
  2- Independent developers or testers within the development teams or the project team.
  3- Independent test team or group within the organization**.**
  4- Independent testers from the business organization or user community, or with specializations in specific test types.
  5- Independent testers external to the organization, either working on-site (insourcing) or off-site (outsourcing).
- **Benefits & Drawbacks of Independence:**

Benefits	Drawbacks
1- Independent testers are likely to recognize different kinds of failures.	1- Isolation from the development team, delays in providing feedback.
	2- Developers may lose a sense of responsibility for quality.
2- Independent testers can verify, or disprove assumptions made by stakeholders.	3- Independent testers may be seen as a bottleneck or blamed for delays in release.
	4- Independent testers may lack some important information.

5.1.2 Tasks of a Test Manager and Tester

Test Manager	Test Manager
1- Test policy - Test Strategy - Test Plan.	1- Review and contribute to test plans.
2- Test monitoring & Control (Test progress report - Test summary report)	2- Assess requirements for testability.
3- Initiate the analysis, design, implementation, and execution of tests.	3- Test conditions, test cases, test procedures, test data, & test execution schedule.
4- Configuration Management.	4- Test Environment setup.
5- Metrices.	5- Test Execution.
6- Tools selection.	6- Test automation.
7- Test Environment Implementation Decision.	7- Non-Functional Testing.
8- Develop the skills and careers of testers.	8- Review tests developed by others.

5.2.1 Purpose and Content of a Test Plan

Test planning activities:

1- Determining the scope, objectives, and risks of testing.
2- Defining the overall approach of testing.
3- Integrating and coordinating the test activities into the software lifecycle activities.
4- 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.
5- Scheduling of test analysis, design, implementation, execution, and evaluation activities.
6- Selecting metrics for test monitoring and control.
7- Budgeting for the test activities.
8- Determining the level of detail and structure for test documentation.

5.2.2 Test Strategy and Test Approach

Strategy	Explanation
Analytical	Based on an analysis of some factor (e.g., requirement or risk). **E.g.** (Risk-based testing) where tests are designed and prioritized based on the level of risk.
Model-based	Test strategy, tests are designed based on model of required aspect of the product, such as a function, a business process, an internal structure, or a non-functional characteristic. **E.g.,** business process models, state models, and reliability growth models.
Methodical	Relies on making systematic use of some predefined set of tests or test conditions, such as a taxonomy of common or likely types of failures or a list of important quality characteristics. **E.g.,** Error Guessing.
Process-complaint (Standard Complaint)	Involves analyzing, designing, and implementing tests based on external rules and standards, such as those specified by industry-specific standards.
Directed (Consultative)	Driven by the advice, guidance, or instructions of stakeholders, business domain experts, or technology experts, who may be outside the test team or outside the organization.
Regression-Averse	Motivated by a desire to avoid regression of existing capabilities. Includes the reuse of existing test-ware, extensive automation of regression tests, and standard test suites.
Reactive	Testing is reactive to the component or system being tested, and the events occurring during test execution, rather than being pre-planned (as the preceding strategies are). **E.g.,** Exploratory Testing.

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

Entry Criteria	Exit Criteria

Define the preconditions for undertaking a given test activity.	Define what conditions must be achieved to declare a test level or a set of tests completed.
Typical entry criteria include: 1- Availability of testable requirements, user stories, and/or models. 2- Availability of test items that have met the exit criteria for any prior test levels. 3- Availability of test environment. 4- Availability of necessary test tools. 5- Availability of test data and other necessary resources.	**Typical exit criteria include:** 1- Planned tests have been executed. 2- A defined level of coverage has been achieved. 3- The number of unresolved defects is within an agreed limit. 4- The number of estimated remaining defects is sufficiently low. 5- The evaluated levels of quality characteristics are sufficient.

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

5.2.4 Test Execution Schedule

Priority for executing test cases is by the **highest priority**. So, if we have two test cases (The first test case priority is 1 and the second test case priority is 3), you should execute the one with the **lowest number** (The first test case with priority 1) **if there are no dependencies**.

5.2.5 Factors Influencing the Test Effort

Product characteristics	Development process characteristics	People characteristics	Test Results
Required level of detail for test documentation	Stability and maturity of the organization	The skills and experience of the people involved, especially with similar projects	The number and severity of defects found
Risks	The development model	Team cohesion and leadership	Amount of rework required
Quality of the test basis	The test approach		
Size - Complexity	The tools used		
Requirements for quality characteristics (e.g., Security)	The test process		
legal and regulatory compliance	Time pressure		

5.2.6 Test Estimation Techniques

The Metrics-based	The expert-based

Estimating the test effort based on metrics of former similar projects or based on typical values.	Estimating the test effort based on the experience of the owners of the testing tasks or by experts.
Burn-down Chart (Agile)	Planning Poker (Agile)
Defect Removal Models (Sequential)	Wideband Delphi (Sequential)

5.3.1 Metrics Used in Testing

Common test metrices:

1- Percentage of planned work done in (**Test case preparation/implementation** - **Test environment preparation**).
2- Test case execution (e.g., **number of test cases run/not run**, **test cases passed/failed**).
3- Defect information (e.g., **defects found and fixed**, **failure rate**, and **confirmation test results**).
4- Test coverage of requirements, user stories, acceptance criteria, risks, or code.
5- Task completion, resource allocation and usage, and effort.
6- Cost of testing.

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.**
- The test report prepared during a test activity may be referred to as a test progress report, while a test report prepared at the end of a test activity may be referred to as a test summary report.
- **Typical test progress reports may also include:**
  - The status of the test activities and progress against the test plan.
  - Factors impeding progress.
  - Testing planned for the next reporting period.
  - The quality of the test object.
- **Typical test progress reports and test summary reports may include:**
  - Summary of testing performed.
  - Information on what occurred during a test period**.**
  - Deviations from plan, including deviations in schedule, duration, or effort of test activities.
  - Status of testing and product quality with respect to the exit criteria or definition of done.
  - Factors that have blocked or continue to block progress.
  - Metrics of **defects**, **test cases**, **test coverage**, **activity progress**, and **resource consumption**.
  - Reusable test work products produced.

5.4 Configuration Management

- The purpose is to **establish and maintain the integrity of the component or system**, the test-ware, and their **relationships to one another** through the project and product lifecycle.
- During test planning, configuration management procedures and infrastructure (tools) identified and implemented.
- **To properly support testing, configuration management may involve ensuring the following:**
  - All items of test-ware 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.

5.5.1 Definition of Risk

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

5.5.2 Product and Project Risks

- **Product Risks (Quality Risks):**
  - Product risk involves the possibility that a work product may fail to **satisfy the legitimate needs of its users and/or stakeholders**, examples include:
    - Software might not perform its intended functions.
    - A system architecture may not adequately support some non-functional requirement(s).
    - A particular computation may be performed incorrectly in some Circumstances.
    - A loop control structure may be coded incorrectly.
    - Response-times may be inadequate for a high-performance transaction processing system.
    - User experience (UX) feedback might not meet product expectations.
- **Project Risks:**
  - Project risk involves situations that, should they occur, may have a **negative effect on a project's ability to achieve its objectives**. Examples of project risks include:
    - **Project issues**: Delays, Inaccurate estimates, Late changes.
    - **Organizational Issues**: Skills, Training, Staff, Personnel issues, Users, Business Staff.
    - **Political Issues**: Communication, Follow up, Improper attitude.
    - **Technical Issues**: Requirements, Test Environment, Data Conversion, Migration planning, Tool Support, Weaknesses in the development process, Poor Defect Management.
    - **Supplier Issues**: Third Party, Contractual issues.

5.6 Defect Management

A defect report filed during dynamic testing typically includes:

- Identifier – Title – Summary – Date – Author – Test Item – Test Environment.
- The development lifecycle phase(s) in which the defect was observed.
- A description of the defect to enable reproduction and resolution, including logs, database dumps screenshots, or recordings (if found during test execution).
- Expected and actual results.
- Scope or degree of impact (severity) of the defect on the interests of stakeholder(s).
- Urgency/priority to fix.
- State of the defect report.
- Conclusions, recommendations, and approvals.
- Global issues, such as other areas that may be affected by a change resulting from the defect.
- Change history.
- References including the test case that revealed the problem.

Chapter 6: Tools Support for Testing

6.1.1 Test Tool Classification

- **Test tools purposes:**

- Improve the efficiency of test activities by automating repetitive tasks or tasks that require significant resources when done manually.
- Improve the efficiency of test activities by supporting manual test activities throughout the test process.
- Improve the quality of test activities by allowing for more consistent testing and a higher level of defect reproducibility.
- Automate activities that cannot be executed manually.
- Increase reliability of testing.

- **Intrusive tools:**
  - Intrusive Tools may affect the actual outcome of the test.
  - The consequence of using intrusive tools is called the probe effect.
- **Tools Support:**

Tool support for	Tools	
Management of Testing and Test-ware	1- Test management tools and Application Lifecycle Management tools (ALM)	3- Requirements management Tools
		4- Configuration management tools
	2- Continuous integration tools (**D**)	5- Defect (Incident) management Tools
Static Testing	1- Tools that support reviews	2- Static analysis tools (**D**)
Test Design and Implementation	1- Test Design tools	2- Model-Based Testing tools
	3- Test data preparation tools	4- Behavior Driven Development (BDD)
	5- Acceptance Test Driven Development (ATDD)	6- Test Driven Development (TDD) tools (**D**)
Test Execution and Logging	1- Test execution tools (e.g., **regression** tests)	2- Coverage tools (e.g., **requirements** coverage, **code** coverage (**D**))
	3- Test harnesses (**D**)	4- Unit test framework tools (**D**)
Performance Measurement and Dynamic Analysis	1- Performance testing tools	2- Monitoring tools
	3- Dynamic analysis tools (**D**)	
Specialized Testing Needs	1- Data quality assessment	2- Data conversion and migration
	3- Usability testing	4- Accessibility testing
	5- Localization testing	6- Security testing
	7- Portability testing (e.g., testing software **across multiple supported platforms**)	

6.1.2 Benefits and Risks of Test Automation

Benefits	Risks
1- Reduction in repetitive manual work, thus saving time.	1- Underestimating time, cost, & effort for the initial introduction of a tool and to achieve significant benefit from it.

2- More objective assessment.	2- Unrealistic expectations of the tool.
3- Greater consistency and repeatability.	3- The effort required to maintain the test assets generated by the tool may be underestimated.
4- Easier access to information about testing.	4- Relationships and interoperability issues between critical tools may be neglected.
	5- The tool may be relied on too much.
	6- Version control of test assets may be neglected.
	7- Vendor problems (**Retire** or **sell** the **tool**/ **poor response**/go **out of business**.
	8- An open-source project may be suspended.
	9- A new platform or technology may not be supported by the tool.
	10- There may be no clear ownership of the tool (e.g., for updates).

6.1.3 Special Considerations for Test Execution and Test Management Tools

Test execution tools

- **Capturing Tools**:
  - Recording the actions of a manual tester seems attractive, but this approach does not scale to large numbers of test scripts.
  - May be unstable when unexpected events occur.
- **Data-driven Testing Approach**:
  - Separates out the test inputs and expected results, usually into a spreadsheet, and uses a more generic test script that can read the input data and execute the same test script with different data.
  - Testers who are not familiar with the scripting language can create new test data for these predefined scripts.
- **Keyword-driven Testing Approach**:
  - Generic script processes keywords describing the actions to be taken (also called action words), which then calls keyword scripts to process the associated test data.
  - Testers (even if they are not familiar with the scripting language) can then define tests using the keywords and associated data, which can be tailored to the application being tested.
- **Model-Based Testing (MBT) Tools**:
  - Enable a functional specification to be captured in the form of a model, such as an activity diagram.
  - This task is generally performed by a system designer.
  - The MBT tool interprets the model to create test case specifications which can then be saved in a test management tool and/or executed by a test execution tool.

6.2.1 Main Principles for Tool Selection

Considerations for Tool Selection

- Assessment of the maturity of the organization, its strengths, and weaknesses.
- Identification of opportunities for an improved test process supported by tools.
- Understanding of the technologies used by the test object(s), to select a tool that is compatible with that technology.

- The build and continuous integration tools are already in use within the organization, to ensure tool compatibility and integration.
- Evaluation of the tool against clear requirements and objective criteria.
- Consideration of whether the tool is available for a free trial period (and for how long).
- Evaluation of the vendor.
- Identification of internal requirements for coaching, mentoring, & training needs.
- Consideration of pros and cons of various licensing models (e.g., **commercial**, or **open source**).
- Estimation of a cost-benefit ratio.
- A proof-of-concept evaluation should be done to see if it works with the organization's infrastructure.

6.2.2 Pilot Projects for Introducing a Tool into an Organization

Introducing the selected tool starts with a pilot project, which has the following objectives:

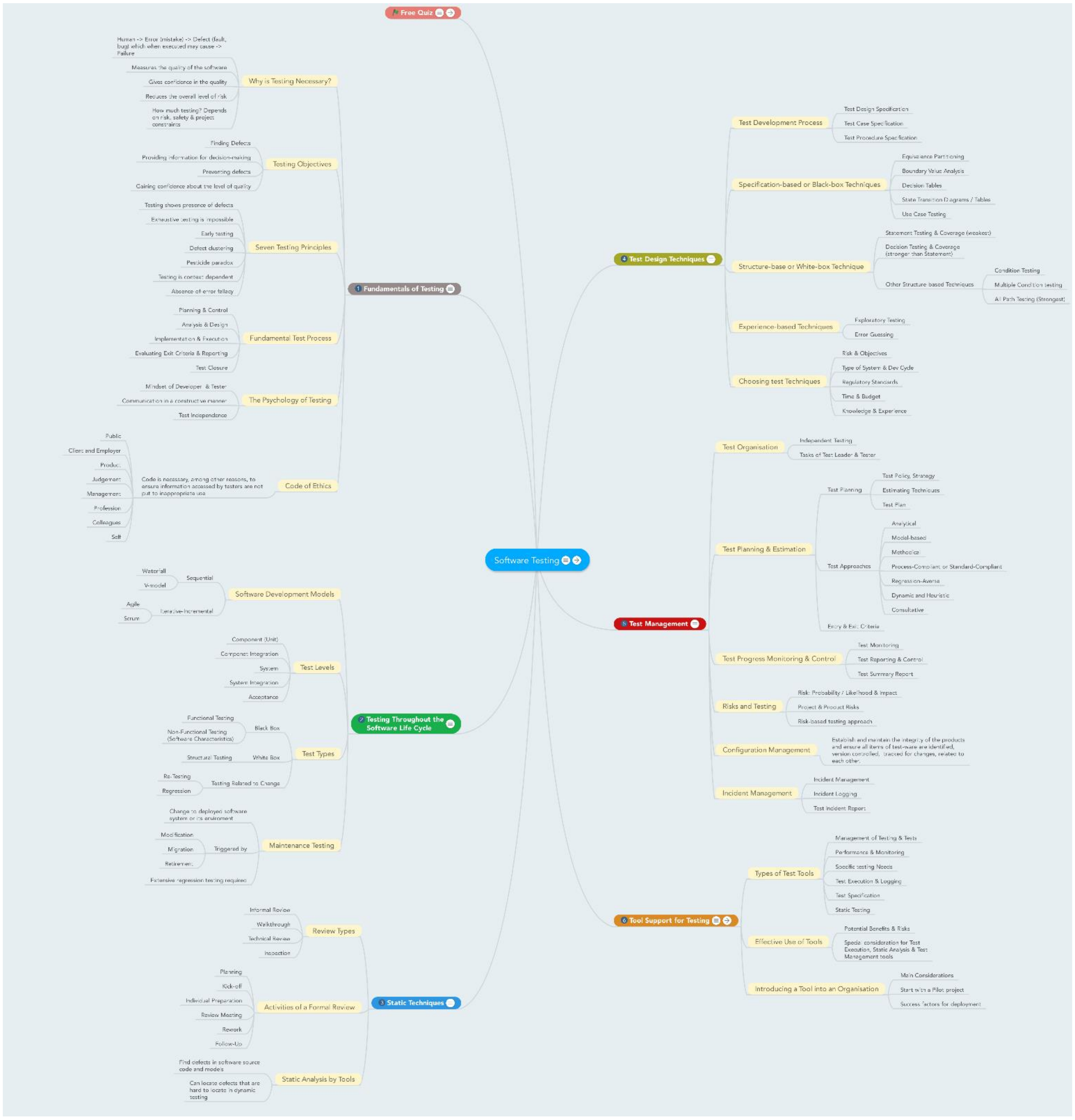
- Gaining knowledge about the tool.
- Evaluating how the tool fits with existing processes and practices.
- Deciding on standard ways of using, managing, storing, and maintaining the tool and the test assets.
- Assessing whether the benefits will be achieved at reasonable cost.
- Understanding the metrics that you wish the tool to collect and report, and configuring the tool to ensure these metrics can be captured and reported.
- Learn -> Evaluate -> Decide -> Assess.

6.2.3 Success Factors for Tools

Success factors of tools within an organization include:

- Rolling out the tool to the rest of the organization incrementally.
- Adapting and improving processes to fit with the use of the tool.
- Providing training, coaching, and mentoring for tool users.
- Defining guidelines for the use of the tool (e.g., **internal standards for automation**).
- Implementing a way to gather usage information from the actual use of the tool.
- Monitoring tool use and benefits.
- Providing support to the users of a given tool.
- Gathering lessons learned from all users.

Cheat Sheet


Software Testing
Free Quiz
1 Fundamentals of Testing
Why is Testing Necessary?
Human -> Error (mistake) -> Defect (fault, bug) which when executed may cause -> Failure
Measures the quality of the software
Gives confidence in the quality
Reduces the overall level of risk
How much testing? Depends on risk, safety & project constraints
Testing Objectives
Finding Defects
Providing information for decision-making
Preventing defects
Gaining confidence about the level of quality
Seven Testing Principles
Testing shows presence of defects
Exhaustive testing is impossible
Early testing
Defect clustering
Pesticide paradox
Testing is context dependent
Absence-of-error fallacy
Fundamental Test Process
Planning & Control
Analysis & Design
Implementation & Execution
Evaluating Exit Criteria & Reporting
Test Closure
The Psychology of Testing
Mindset of Developer & Tester
Communication in a constructive manner
Test Independence
Code of Ethics
Code is necessary, among other reasons, to ensure information accessed by testers are not put to inappropriate use
Public
Client and Employer
Product
Judgement
Management
Profession
Colleagues
Self
2 Testing Throughout the Software Life Cycle
Software Development Models
Sequential
Waterfall
V-model
Iterative-Incremental
Agile
Scrum
Test Levels
Component (Unit)
Component Integration
System
System Integration
Acceptance
Test Types
Black Box
Functional Testing
Non-Functional Testing (Software Characteristics)
White Box
Structural Testing
Testing Related to Change
Re-Testing
Regression
Maintenance Testing
Change to deployed software system or its environment
Triggered by
Modification
Migration
Retirement
Extensive regression testing required
3 Static Techniques
Review Types
Informal Review
Walkthrough
Technical Review
Inspection
Activities of a Formal Review
Planning
Kick-off
Individual Preparation
Review Meeting
Rework
Follow-Up
Static Analysis by Tools
Find defects in software source code and models
Can locate defects that are hard to locate in dynamic testing
4 Test Design Techniques
Test Development Process
Test Design Specification
Test Case Specification
Test Procedure Specification
Specification-based or Black-box Techniques
Equivalence Partitioning
Boundary Value Analysis
Decision Tables
State Transition Diagrams / Tables
Use Case Testing
Structure-base or White-box Technique
Statement Testing & Coverage (weakest)
Decision Testing & Coverage (stronger than Statement)
Other Structure-based Techniques
Condition Testing
Multiple Condition testing
All Path Testing (Strongest)
Experience-based Techniques
Exploratory Testing
Error Guessing
Choosing test Techniques
Risk & Objectives
Type of System & Dev Cycle
Regulatory Standards
Time & Budget
Knowledge & Experience
5 Test Management
Test Organisation
Independent Testing
Tasks of Test Leader & Tester
Test Planning & Estimation
Test Planning
Test Policy, Strategy
Estimating Techniques
Test Plan
Test Approaches
Analytical
Model-based
Methodical
Process-Compliant or Standard-Compliant
Regression-Averse
Dynamic and Heuristic
Consultative
Entry & Exit Criteria
Test Progress Monitoring & Control
Test Monitoring
Test Reporting & Control
Test Summary Report
Risks and Testing
Risk: Probability / Likelihood & Impact
Project & Product Risks
Risk-based testing approach
Configuration Management
Establish and maintain the integrity of the products and ensure all items of test-ware are identified, version controlled, tracked for changes, related to each other.
Incident Management
Incident Management
Incident Logging
Test Incident Report
6 Tool Support for Testing
Types of Test Tools
Management of Testing & Tests
Performance & Monitoring
Specific testing Needs
Test Execution & Logging
Test Specification
Static Testing
Effective Use of Tools
Potential Benefits & Risks
Special consideration for Test Execution, Static Analysis & Test Management tools
Introducing a Tool into an Organisation
Main Considerations
Start with a Pilot project
Success factors for deployment