Software Testing and Quality Assurance

Chapter 1: Introduction

Definition

- Testing is the process of exercising or evaluating a system or system components by manual or automated means to verify that it satisfies specified requirements. (IEEE)
- Software testing is the process of executing a program or system with the intent of finding errors. (Myers)
- It involves any activity aimed at evaluating an attribute or capbility of a program or system and determining that it meets it's required results. (Hetzel)

What's NOT Software Testing?

- Demonstrating that errors aren't present.
- Showing that a program performs it's intended functions correctly.
- Establishing confidence that a program does what is supposed to do.

Debugging vs Testing

- `Debugging` is removing errors from your programs. `Testing` is locating undiscovered errors.

Static and Dynamic testing

- `Static testing` is checking whether the SRS is as per user requirements or not. Includes techniques of code reviews, code inspections, walkthroughs and software technical reviews (STRs).
- `Dynamic testing` is the actual execution of the code when the code is ready even a unit or module is ready. Includes black-box, white-box and gray-box testing techniques.

Levels of Software Testing

- **Debug**: The successful correction of a failure.
- **Demonstrate**: Process of showing that major features work with typical input.
- **Verify**: Process of finding as many faults in the application under test (AUT) as possible.
- **Validate**: Process of finding as many faults in requirements, design and AUT.
- **Prevent**: Avoiding errors in development of requirements, design and implementation by self-checking techniques.
- `Prevent -> Validate -> Verify -> Demonstrate -> Debug`

Positive vs Negative Testing

Positive testing	Negative testing
Test for normal or common workflows.	Test for abnormal operations.
Test with proper variety of legal test data.	Test with illegal or invalid values.
Does the application function correctly?	Does the program do what it should not do?

What's the positive view of negative testing?

- The job of testing is to discover errors before the user does.
- Mentality of the tester has to be destructive opposite to that of the developer which should be constructive.

Software Verification and Validation

- `Software Verification`
    - It's process of evaluating a system or component to determine whether the products of a given development phase satisfy the condition imposed at the start of the phase.
    - It's process of evaluating, reviewing, inspecting and doing desk checks of work products such as requirement, design and code specification.
    - Human testing activity as it involves looking at the documents on paper.
- `Software Validation`
    - It's process of evaluating a system or component during or at the end of the development process whether to determine it satisfies the specified requirements or not.
    - It's involves executing the actual software.

Why should we test?

- The Technical case
- The Business case
- The Professional case
- The Economics case
- To improve quality
- For verification and validation
- For reliability estimation

Who should do testing?

- Everyone involved in the SDLC is responsible to perform testing.

How much should we test?

- If multiple testing of the functionality of the software not showcasing any defects, then it's the right time to stop testing the software.
- If we found defects with a small potion of overall functionality, we should continue testing.

Why is designing good test case is hard?

- Different types of test cases are needed for different classes of information.
- All test cases within a test suite will not be good.
- People create test cases according to certain testing styles.

Test cases

- Brief statement of something that should be tested. (Brain Marick)
- The best test cases are the ones that find bugs. (Cem Kaner)

Measurement of Testing

- There is no single scale that is avaliable to measure the testing progress. But metrics can be computed at the organizational, process, project and product levels.

Incremental Testing Approach

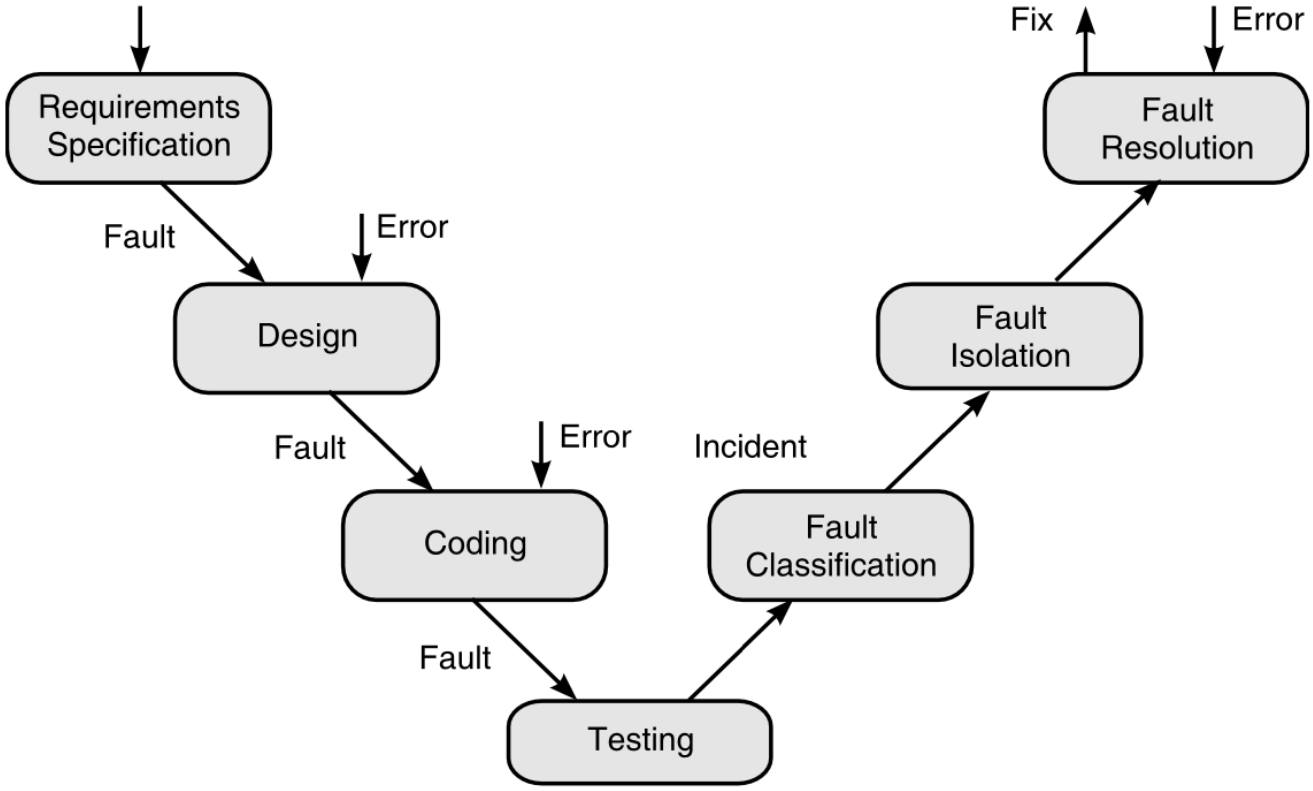
- Tester should be familiar with software testing techniques and application under test.
- Basic stages
  - `Exploration` : gain amiliarity with the application.
  - `Baseline test` : devise and execute a simple test case.
  - `Trends analysis` : evaluate whether the application performs as expected when actual output can't be predetermined.
  - `Inventory` : identify the different categories of data and create a test for each item.
  - `Inventory combinations` : combine different input data.
  - `Push the boundaries` : evaluate application behavior at data boundaries.
  - `Devious data` : evaluate system response when specifying bad data.
  - `Stress the environment` : attempt to break the system.

Basic terminologies related to testing

- **Error(bug or mistake)**: mistake during SDLC.
- **Fault(Defect)**: missing or incorrect statement in a program resulting from an error. Representation of an error.
- **Failure**: the manifested inability of a system or component to perform a required function within a specified limits.
- **Incident**: the symptom associated with a failure that alerts the user to the occurrence of a failure.

- **Test**: act of exercising software with test cases.
- **Test case**: set of inputs and a list of expected outputs.
  - Inputs
    - `Preconditions` : circumstances that hold prior to test case execution.
    - `Actual inputs` : inputs identified by some testing method.
  - Outputs
    - `Post conditions`
    - `Actual outputs`
- **Test suite**: collection of test scripts that is used for validating big fixes (finding new bugs) within a logical or physical area of the product.
- **Test script**: stey by step instructions that describe how a test case is to be executed.
- **Test ware**: all of testing documentation created during the testing process.
- **Test oracle**: any means used to predict the outcome of a test.
- **Test log**: chronological record of all relevant details about the execution of a test.
- **Test report**: document describing the conduct and results of testing carried out for a system.

Testing life cycle



When to stop testing?

- The pessimstic approach is to stop testing whenever some or any of the allocated resources are exhausted.
- The optimistic approach is to stop testing when either the reliability meets the requirement, or the benefit of continue testing can’t justify the testing cost.(Yang)

Principles of Testing

- Testing should be based on user requirements.
- Testing time and resources are limited.
- Exhaustive testing impossible.
- Use effective resources to test.
- Test planning should be done early.
- Testing should begin “in small” and progress toward testing “in large”.
- Testing should be conducted by an independent third party.
- All tests should be traceable to customer requirements.
- Assign best people for testing (Avoid programmers).
- Test should be planned to show software defects and not their absence.
- Prepare test reports including test cases and test results to summarize the result of testing.
- Advance test planning is a must and should be updated in a timely manner.

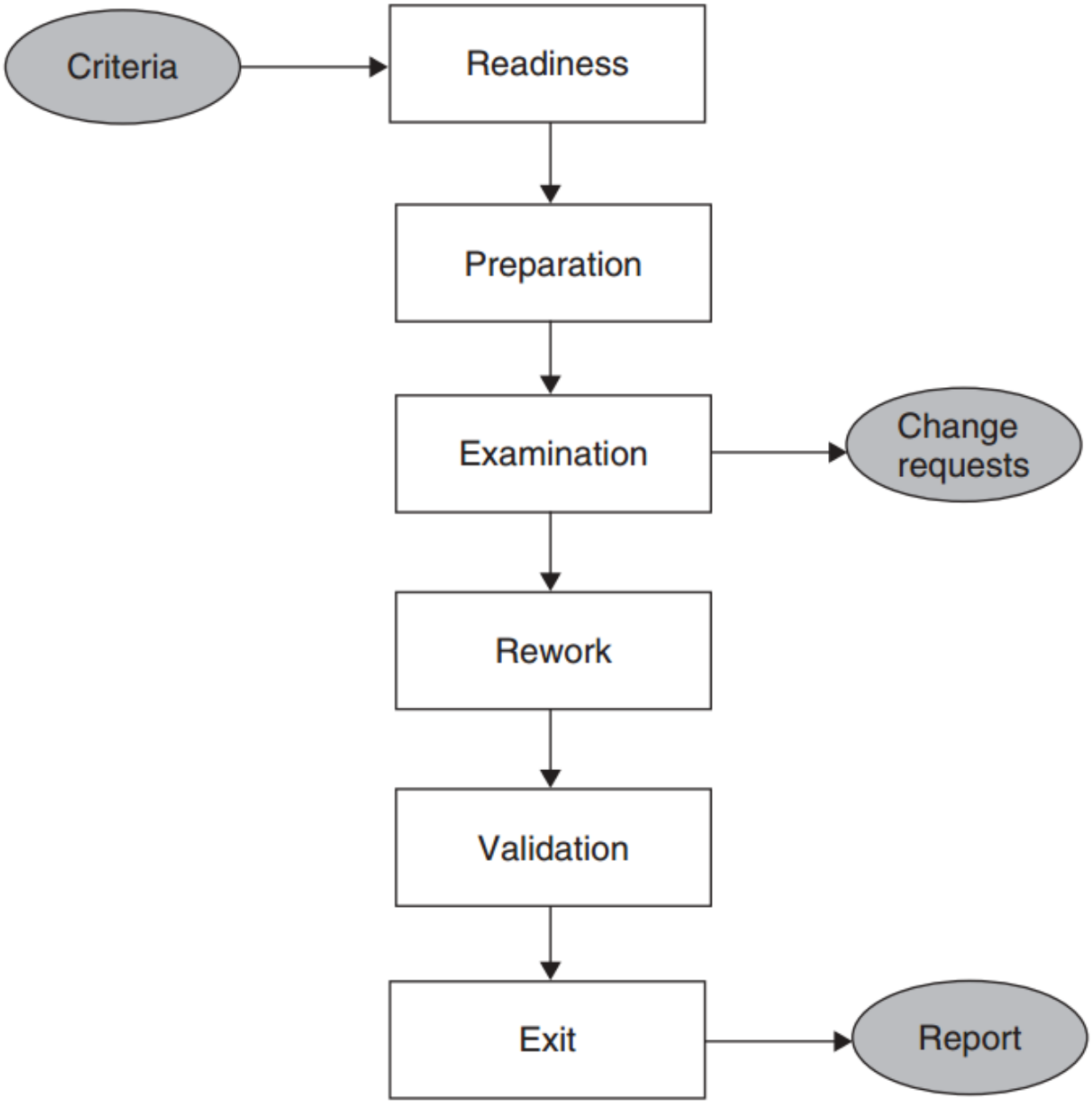
Limitations of testing

- Testing can show the presence of errors not their absence.
- We will never find the last bug in any application.
- The domain of possible inputs is to large to test.
- There are too many possible paths through the program to test.
- Various testing techniques are complemntary in nature.

Chapter 2: Levels of Software Testing

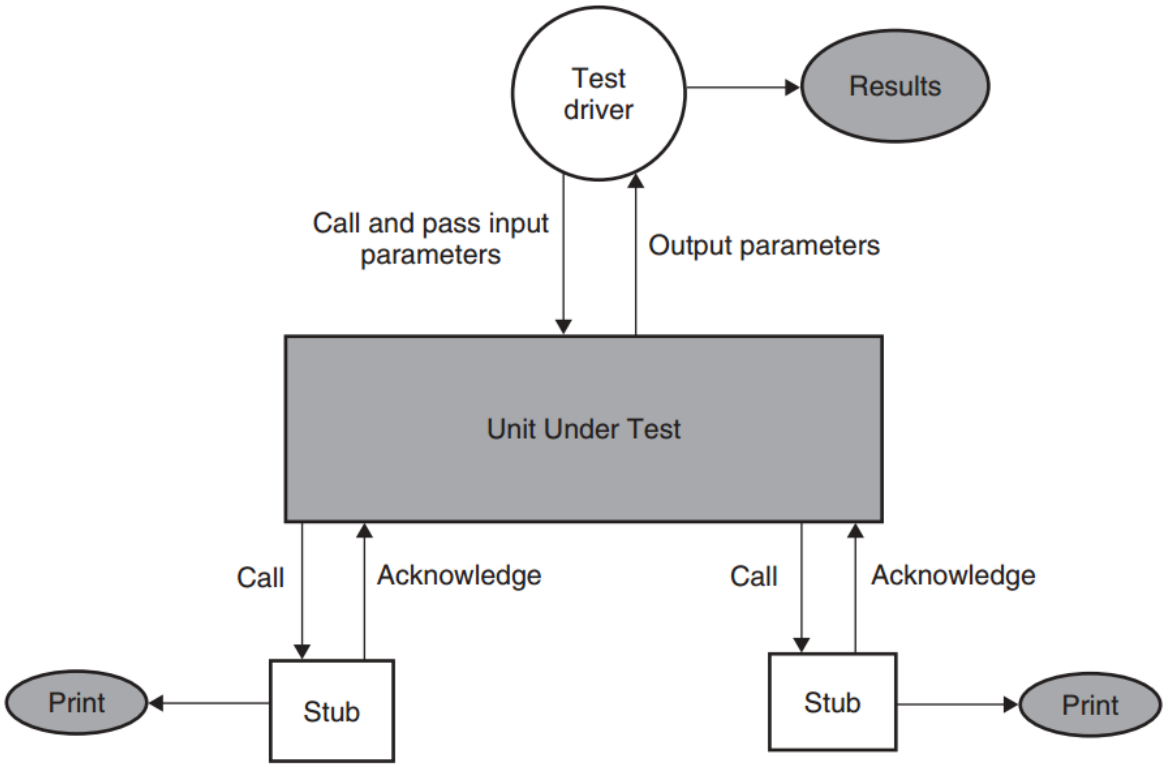
Unit Testing

- Testing program units in isolation.
- `Unit` are functions, procedures, methods or a class.
- Activities:
  - Execute every line of code.
  - Execute every predicate in the unit to evaluate them true and false separately.
  - Observe that the unit performs its inteded function and ensure that it contains no known errors.
- Phases:
  - **Static unit testing:** code is reviewed by applying techniques commonly known as inspection and walkthrough.



- **Dynamic unit testing:** a program unit is actually executed in isolation. Emulates the context of (test driver and stubs) the unit under test.
  - `Test driver` : is a program that invokes the unit under test with input values.

- `Stub` : dummy subprogram that replaces a unit that is called by unit under test.

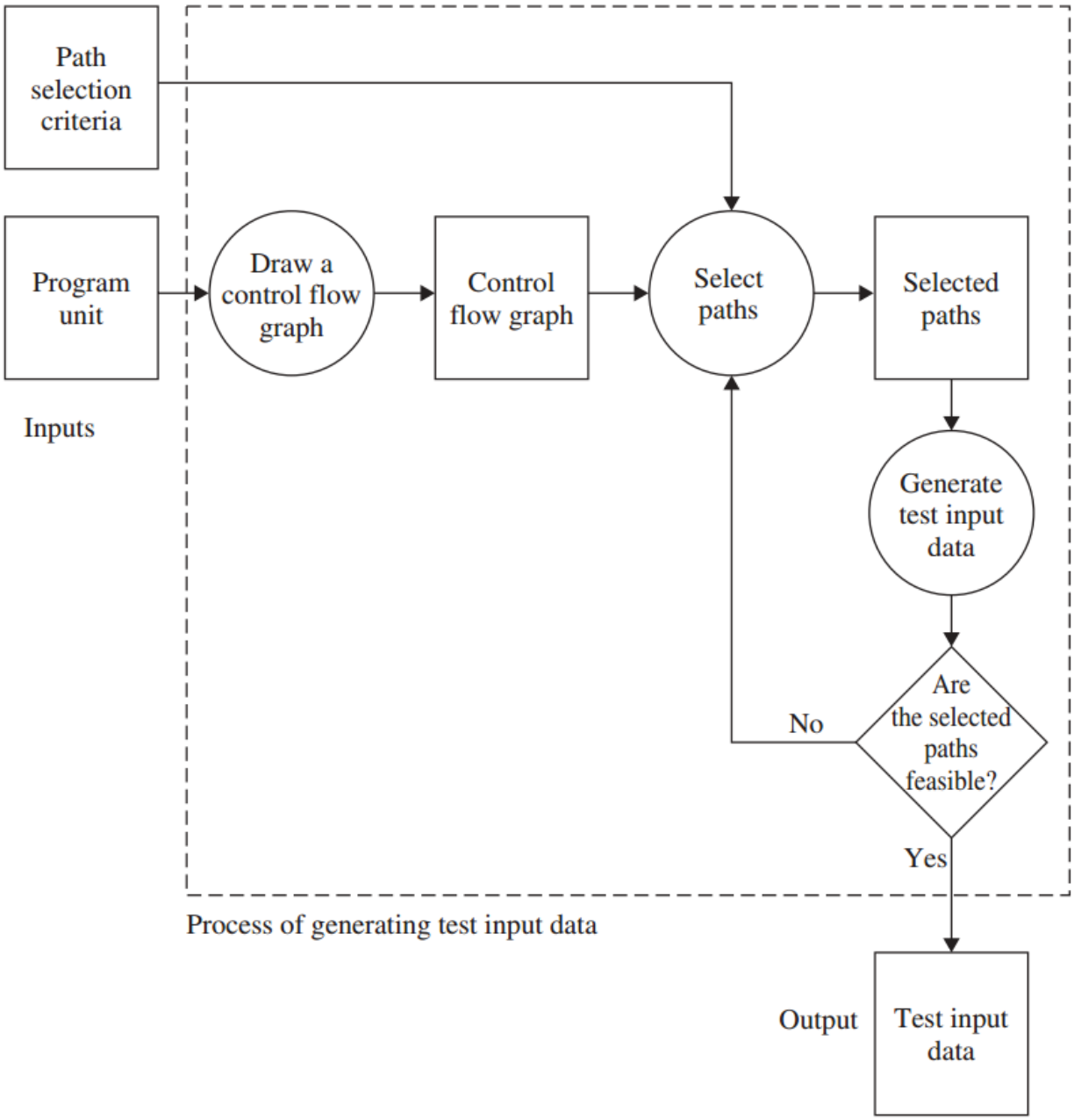


- Built in tracking mechanisms and exceptions can be used as defect prevention mechanisms.

Control flow Testing

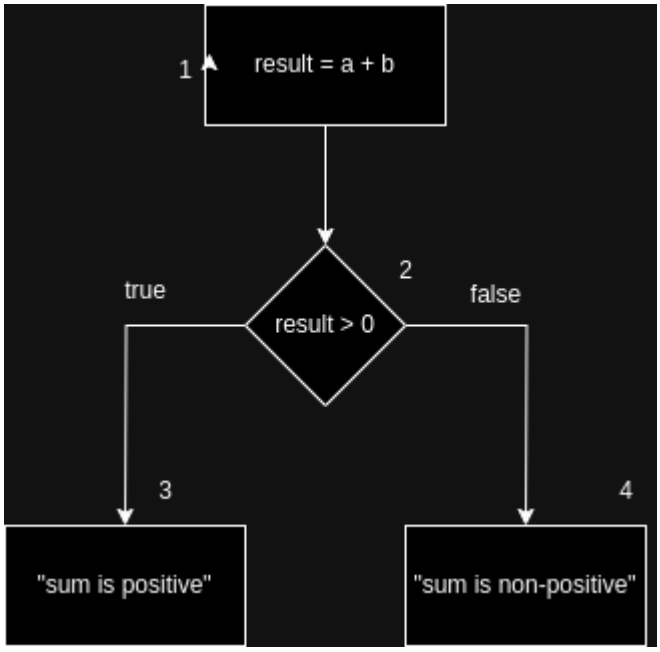
- `Control flow` refers to a code that define sequence, decision function or a class of a program.
- `Control flow testing` is a kind of structural testing, which is performed by programmers to test code written by them. The main idea is to appropriately select a few paths in a program unit and observe whether or not the selected paths produce the expected outcome.
- A control flow graph (CFG) is a graphical representation of a program unit.
- Steps:

- Input (src code, path selection criterion) -> CFG -> Selection of Paths -> Output



- Paths:
  - A `path` is represented as a sequence of computation and decision nodes from the entry node to the exit node.
  - Path selection criteria:
    - Select all paths:
      - All defects can be detected.

```
def sum(a, b):
    result = a + b

    if result > 0:
        print("Sum is positive.")
    else:
        print("Sum is non-positive.")
```



- Path 1: 1 -> 2(T) -> 3
- Path 2: 1 -> 2(F) -> 4

- Statement Coverage Criterion:
  - Executing individual program statements and observing the outcome.
  - Complete statement coverage is the weakest coverage criterion in program testing.
  - Any test suite that achieves less than statement coverage for new software is considered to be unacceptable.

```
def sum(a, b):
    result = a + b

    if result > 0:
        print("Sum is positive.")
    else:
        print("Sum is non-positive.")

# test case covering the first condition
sum(3, 4)

# test case covering the second condition
sum(-2, 2)
```

- Branch Coverage Criterion:
  - Covering a branch means selecting a path that includes the outgoing edge.

```
def sum(a, b):
    result = a + b

    if result > 0:
        print("Sum is positive.")
    else:
        print("Sum is non-positive.")

# test case covering the true branch
sum(3, 4)

# test case covering the false branch
sum(-2, 2)
```

  - Predicate Coverage Criterion:
    - If all possible combinations of truth values of the conditions affecting a selected path have been explored under some tests, then we say that predicate coverage is satisfied.

```
def sum(a, b):
    result = a + b

    if result > 0:
        print("Sum is positive.")
    else:
        print("Sum is non-positive.")
```

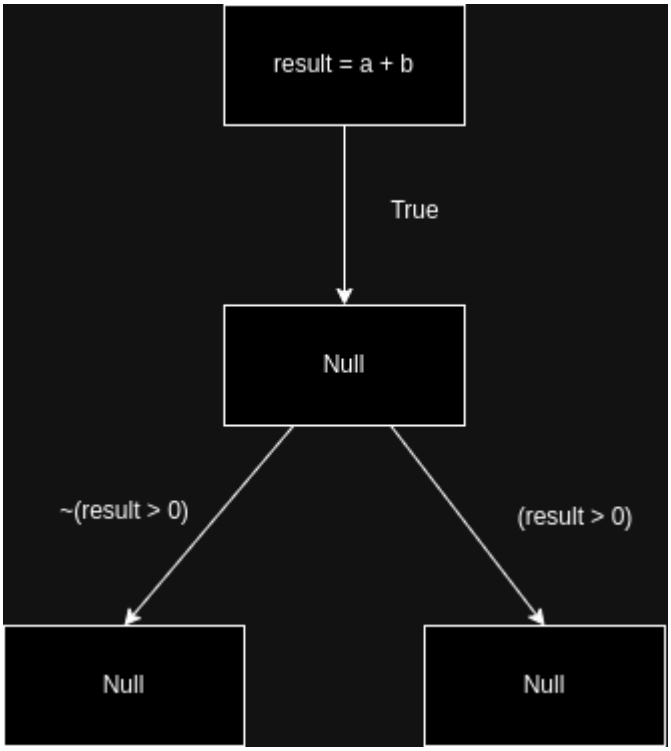
    - Possible combination of truth values = (true, false)
- `Infeasible path` : is a path that is impossible to achieve with any set of inputs.

Data flow Testing

- Using testing paths that include pairs of definition and use use of variables in programs.
- Data Flow Anomaly:
  - `Defined and Defined Again` : assigning two variables without using the first value.
  - `Undefined but Referenced` : use of value of a variable before assigning a value to the variable.
  - `Defined but Not Referenced` : assigning a value to a variable and not using it.
- Data Flow Diagram(DFG):
  - Diagram that describes data definitions and their uses.

- Occurrence of data variables:
  - Definition
  - Undefinition or kill
  - Use:
    - Computation use (c-use)
    - Predicate use (p-use)
- A data flow graph is a directed graph constructed as follows:
  - A sequence of definitions and c-uses is associated with each node of the graph.
  - A set of p-uses is associated with each edge of the graph.
  - The entry node has a definition of each parameter and each nonlocal variable which occurs in the subprogram.
  - The exit node has an undefinition of each local variable.

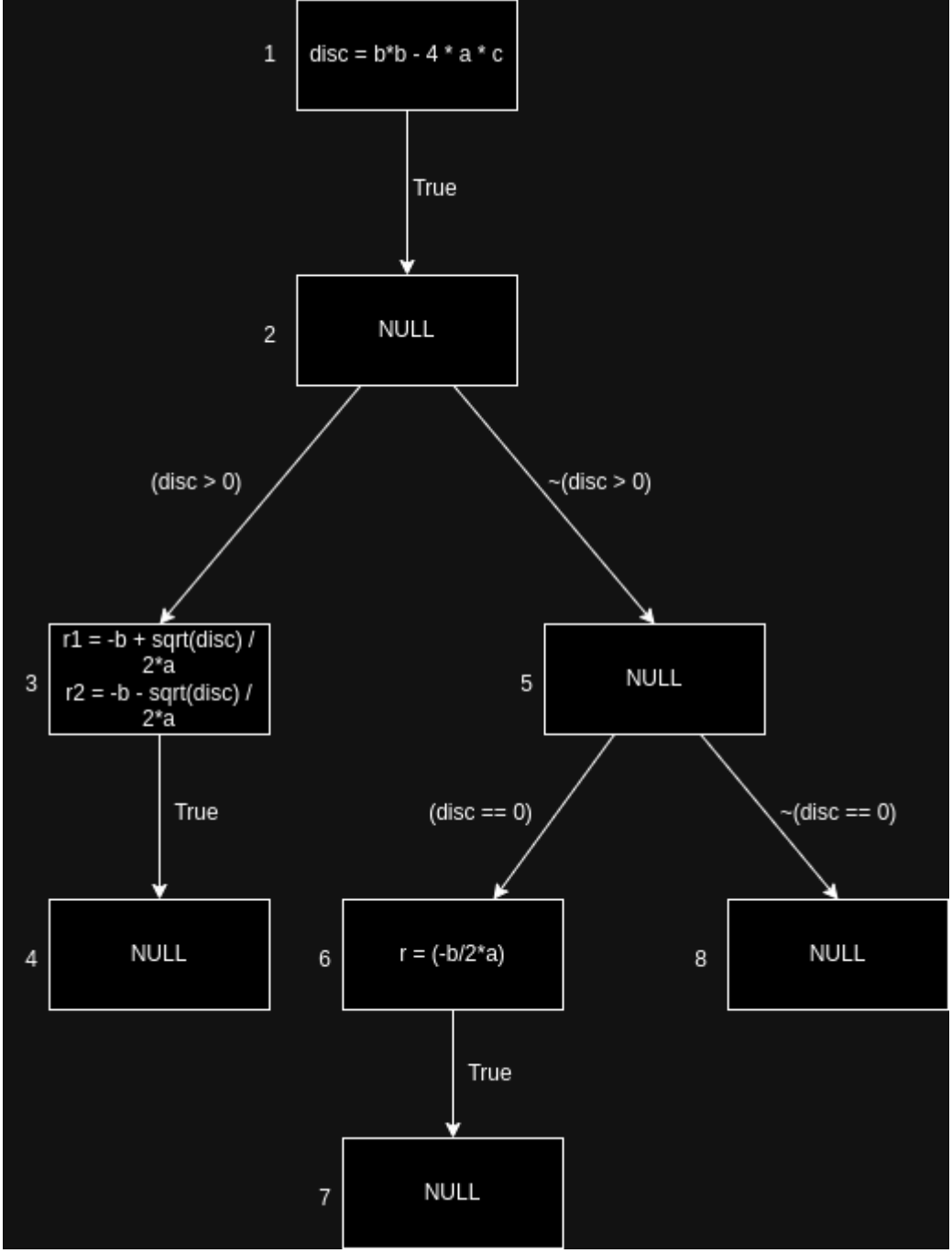
```
def sum(a, b):
    result = a + b

    if result > 0:
        print("Sum is positive.")
    else:
        print("Sum is non-positive.")
```



- Data flow terms:
  - `Global c-use` : a variable x in node i is said to be a global c-use if x has been defined before in a node other than node i.
  - `Definition Clear Path` : is a sub-path where x is not defined at any of the nodes in p.

- `Global definition` : A node i has a global definition of a variable x if node i has a definition of x and there is a def-clear path with respect to x from node i to some node containing a global c-use or edge containing a p-use of variable x.
- `Simple path` : is a path in which all nodes, except possibly the first and the last, are distinct.
- `Du-path` : A path is a definition-use path (du-path) with respect to variable x if node n1 has a global definition of x and either node nk has a global c-use of x and is a def-clear simple path w.r.t. x or edge has a p-use of x and def-clear, loop-free path w.r.t. x.

```
def get_root(a, b, c):
    disc = (b*b) - 4 * a * c
    if disc > 0:
        r1 = (-b + sqrt(disc))/(2 * a)
        r2 = (-b - sqrt(disc))/(2 * a)
        print(r1, r2)
    elif disc == 0:
        r = -b / (2 * a)
        print(r)
    else:
        print("no real roots")
```



Nodes	def(i)	c-use(i)
1	{ disc }	{ a, b, c }
2	{}	{}
3	{ r1, r2 }	{ a, b, c }
4	{}	{}
5	{}	{}
6	{ r }	{ a, b }
7	{}	{}
8	{}	{}

Edges	Predicates	p-use(i)
(1, 2)	True	{}
(2, 3)	(disc > 0)	{ disc }
(3, 4)	True	{}
(2, 5)	~(disc > 0)	{ disc }
(5, 6)	(disc == 0)	{ disc }
(6, 7)	{}	{}
(5, 8)	~(disc == 0)	{ disc }

- Data flow Testing criterias:
  - `All-defs` : For each variable x and for each node i such that x has a global definition in node i , select a complete path which includes a def-clear path from node i to node j having a global c-use of x or edge (j ,k) having a p-use of x.
  - `All-c-uses` : For each variable x and for each node i , such that x has a global definition in node i , select complete paths which include def-clear paths from node i to all nodes j such that there is a global c-use of x in j .
  - `All-p-uses` : For each variable x and for each node i such that x has a global definition in node i , select complete paths which include def-clear paths from node i to all edges (j ,k) such that there is a p-use of x on edge (j ,k).
  - `All-p-uses/Some-c-uses`
  - `All-c-uses/Some-p-uses`
  - `All-uses`
  - `All-du-paths`
- Data flow Testing is good at getting defects compared to Control flow testing.

Domain Testing

- `Input domain` : set of all input values for which the program performs the same computation for every memeber of the set.
- `Program path` : a sequence of instructions from the start of the program to some point of interest in the program.

- Domain errors:
    - **Computation Error:** occurs when a specific input data causes the program to execute the correct, i.e., desired path, but the output value is wrong.
    - **Domain Error:** occurs when an incorrect path is selected by a program if there is a fault in one or more of the conditional statements in the program.
    - Flow graph based tests select paths to satisfy certain coverage criteria but domain testing define a category of faults called domain errors by selecting test data to detect it.
    - Steps:
        - Draw a control flow graph from the given source code.
        - Find all possible interpretations of the predicates.
        - Analyze the interpreted predicates to identify domains.
    - Domains and its boundaries:
        - `Closed Boundary` : A boundary is said to be closed if the points on the boundary are included in the domain of interest.
        - `Open Boundary` : A boundary is said to be open if the points on the boundary do not belong to the domain of interest.
        - `Closed Domain` : A domain is said to be closed if all of its boundaries are closed.
        - `Open Domain` : A domain is said to be open if some of its boundaries are open.
        - `Extreme Point` : An extreme point is a point where two or more boundaries cross.
        - `Adjacent Domains` : Two domains are said to be adjacent if they have a boundary inequality in common.
    - Types of Domain Errors:
        - `Closure Error` : A closure error occurs if a boundary is open when the intention is to have a closed boundary, or vice versa.
        - `Shifted-Boundary Error` : A shifted-boundary error occurs when the implemented boundary is parallel to the intended boundary.
        - `Tilted-Boundary Error` : If the constant coefficients of the variables in a predicate defining a boundary take up wrong values, then the tilted-boundary error occurs.
    - Test selection Criterion
        i. Closed inequality boundary
            - Boundary shift resulting in a reduced domain
            - Boundary shift resulting in an enlarged domain
            - Boundary tilt
            - Closure error

ii. Open inequality boundary
    - Boundary shift resulting in a reduced domain
    - Boundary shift resulting in an enlarged domain
    - Boundary tilt
    - Closure error

iii. Equality boundary

System Integration Testing

- `Module (Component)` : a self-contained element of a system.
- `System` : a collection of modules interconnected in a certain way to accomplish a tangible objective.
- `System integration testing` : a systematic technique for assembling a software system while conducting tests to uncover errors associated with interfacing.
- Integration testing is said to be complete when the system is fully integrated together, all the test cases have been executed, all the severe and moderate defects found have been fixed, and the system is retested.
- Modules are interfaced with other modules to realize the system's functional requirements.
- Types of interfaces:
    - `Procedure Call Interface` : the caller passes on control to the called module.
    - `Shared Memory` : A block of memory is shared between two modules.
    - `Message Passing Interface` : One module prepares a message by initializing the fields of a data structure and sending the message to another module.
- Interface Errors:
    - Construction
    - Inadequate Functionality
    - Location of Functionality
    - Added Functionality
    - Misuse of Interface
    - Misunderstanding of Interface
- Advantages of Integration testing:
    - Defects are detected early
    - It's easier to fix defects detected earlier
    - We get earlier feedback on the health and acceptability of the individual modules and the overall system.
    - Scheduling of defect fixes is flexible, and it can overlap with development.

- Granularity of System Integration Testing:
  - **Intra-system testing:** focuses on testing within a single system or subsystem. It ensures that the internal components of a system work together as expected.
  - **Inter-system testing:** involves testing the interactions between different systems or subsystems. It focuses on verifying that multiple systems can work together as intended.
  - **Pairwise-system testing:** involves testing all possible pairs of system modules.
- Techniques:
  - `Incremental` : a series of test cycles created to add new modules in existing build.
  - `Top-Down Testing` :
    - Testing begins with the high-level modules and progressively moves towards the lower-level modules. The higher-level modules are tested first, and then the testing proceeds to the lower-level modules.
    - Advantages:
      - Early testing of the main functionalities.
      - Helps identify and address major issues early in the development process.
      - Useful when a stable and well-defined high-level design is available.
    - Challenges:
      - Dependencies on lower-level modules might cause delays if they are not ready.
      - Stubs need to be created for testing, which may not accurately represent the lower-level modules.
  - `Bottom-Up Testing` :
    - Testing starts with the lower-level modules, and the integration process moves upwards towards the higher-level modules. The focus is on testing individual components first and then integrating them into larger subsystems and the complete system.
    - Advantages:
      - Early detection of issues in lower-level modules.
      - Allows for incremental testing, making it easier to identify and fix problems in individual components.
      - Suitable when detailed low-level design is available early in the development process.
    - Challenges:
      - Main functionalities may be tested later in the process.
      - Requires the development of drivers for testing, which may not accurately represent the higher-level modules.
  - `Sandwich testing` : hybrid of Top-Down and Bottom Up.
  - `Big Bang` : all the modules tested individually then, all those modules are put together to construct the entire system which is also tested as a whole.

White box vs Black box testing

White box	Black box
Testing technique where the tester has full knowledge of the internal workings, structure, and code of the software being tested.	testing technique where the tester does not have any knowledge of the internal code or implementation details of the software being tested.
The primary objective is to ensure that all code paths are tested and that the internal logic of the software functions as intended.	The primary objective is to validate the software's functionality against its specifications and requirements without considering the internal code structure.
Developers or testers with programming knowledge are typically involved in white box testing.	Testers who may not have programming knowledge and approach the software as a "black box" are involved in black box testing.

System Testing

- Testing that establish whether an implementation conforms to the requirements specified by the customers or not.
- `Basic tests` provide an evidence that the system can be installed, configured, and brought to an operational state.
- `Functionality tests` provide comprehensive testing of requirements within the capabilities of the system.
- `Robustness tests` determine how well the system recovers from various input errors and other failure situations.
- `Interoperability tests` determine whether the system can interoperate with other third-party products.
- `Performance tests` measure the throughput and response time, under various conditions.
- `Scalability tests` determine user scaling, geographic scaling, and resource scaling.
- `Stress tests` put a system under stress in order to determine the limitations of a system and, when it fails, to determine the manner in which the failure occurs.
- `Regression tests` determine that the system remains stable as it cycles through the integration of other subsystems and through maintenance tasks.

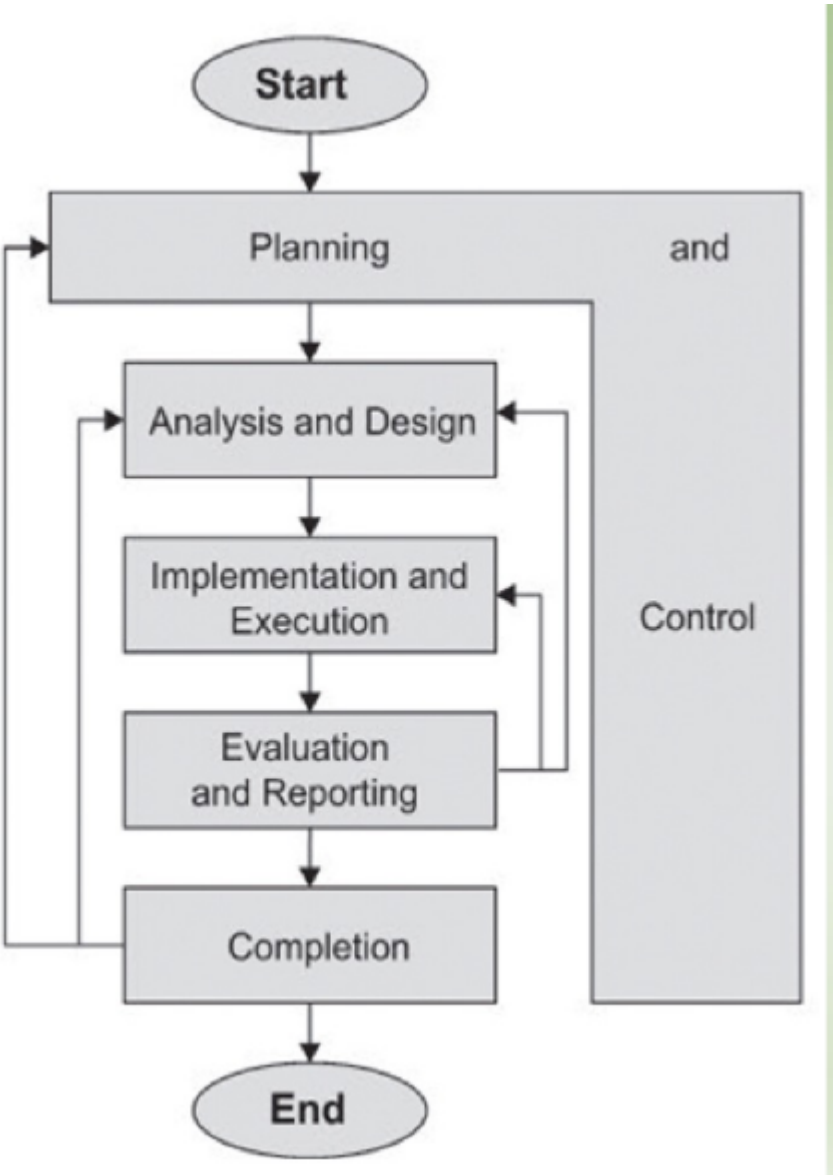
Acceptance Testing

- Acceptance testing is a phase in the software testing process where the software is evaluated to determine whether it satisfies the specified business requirements and is ready for deployment.
- `User Acceptance Testing (UAT)` : conducted by end-users or representatives of the end-users to validate whether the software meets their business requirements.

- `System Acceptance Testing (SAT)` : focuses on ensuring that the entire system, including its hardware, software, and infrastructure, is ready for production deployment.

Chapter 3: Test Management

Fundemental Test Process



Test organization and Independence

- **Adavntages**:
  - Independent testers see other and different defects.
  - Independent testers are unbiased.
  - An independent tester can verify assumptions people made during specification and implementation of the system.
- **Disadvantages**:
  - Isolation from the development team.
  - Independent testers may be the bottleneck as the last checkpoint.
  - Developers may lose a sense of responsibility for quality.

Test leader and Tester

- **Test leader**:
  - Test manager/test coordinator.
  - Plans, monitors and controls the testing activities and tasks.
  - Tasks:
    - Coordination of the test strategy and plan with project managers.
    - Understand the test objectives and risks.
    - Initiate the specification, preparation, implementation and execution of tests.
    - Adapt planning based on test results and progress and take any action to compensate for problems.
    - Set up adequate configuration management for traceability.
    - Introduce metrics for measuring test progress and evaluating the quality of testing and product.
    - Decide what should be automated, to what degree and how.
    - Select tools to support testing and organize trainings for tool users.
    - Decide about the implementation of the test environment.
- Tester:
  - Reviews and contributes to test plan, analyzes, designs, prepares, implements and executes tests.

  - Tasks:

    - Review and contribute to test plans.
    - Analyze review and assess user requirements, specifications and models for testability.
    - Create test specifications.
    - Set up test environment.
    - Prepare and acquire test data

    - Testing process:
        - Implement tests on all test levels
        - Execute and log the tests
        - Evaluate the results
        - Document the deviations from expected results
    - Use test tools
    - Automate tests
    - Measure performance of components and systems if possible
    - Review tests developed by others

Skill required for people involved in testing

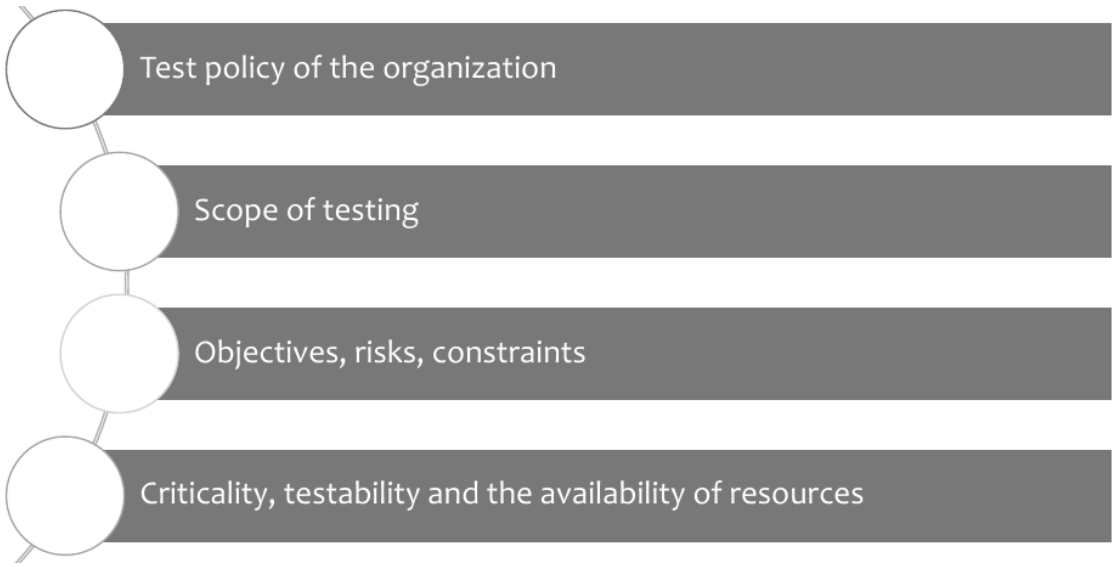
- Application or business domain knowledge
- Technology
- Testing
- Literacy
- Ability to prepare and deliver written and verbal reports
- Ability to communicate efficetively

Test plan

- Project plan for the testing work to be done.
- Documented in a project or master test plan and in separate test plans for test levels such as integrating testing, system testing and acceptance testing.

Test planning

- Continous activity performed in all life cycle processes and activities.



- **Activities**:
  - Determining the scope and risks of testing
  - Identifying the objectives of testing
  - Defining overall approach of testing
  - Integrating and coordinating the testing activities into the software life cycle activities such as `acquisition and supply`, `development`, `operation and maintenance`.
  - Making decision about `what to test`, `what roles will perform the test activities`, `how the test activities should be done`, and `how the test results will be evaluated`.

Entry/Exit criteria

- **Entry criteria**:
  - When to start testing
  - Criterias:
    - Test env't avaliability and readiness
    - Test tool readiness in the test env't
    - Testable code avaliability
    - Test data avaliability
- **Exit criteria**:
  - When to stop testing
  - `Thoroughness measures` : Code coverage and functionality coverage risk.
  - `Estimates` : defect density, reliability measures.
  - `Cost`
  - `Residual risks` : defects not fixed, lack of test coverage in some areas.
  - `Schedule` : time to market.

Test estimation

- A management activity which approximates how long a Task would take to complete.
- **Techniques**:
  - `The metrics-based approach` : estimating the testing effort based on metrics of former or similar projects or based on typical values.
  - `The expert-based approach` : Estimating the tasks by the owner of these tasks or by experts.
- A good solution is to combine the two strategies:
  - First: create the work-breakdown structure and a detailed bottom-up estimate.
  - Second: We then apply models and rules of thumb to check and adjust the estimate bottom-up and top-down using past history.

Factors affecting the test effort

- **Product factors**:
  - The quality of the specification (the test basis)
  - The size of the product
  - The complexity of the problem domain
  - The importance of non functional quality e.g. usability, performance, security etc.
- **Process factors**:
  - The development model
  - Availability of test tools (e.g. test executing tools)
  - Skills of the people involved
  - Time pressure
- **The outcome of testing**:
  - The number of defects
  - The amount of rework required

Test approaches and startegies

- The implementation of test strategy for a specific project.
- **Approaches**:
  - `Preventive approach` : tests are designed as early as possible.
  - `Reactive approach` : tests are designed after the software or system has been produces.
  - `Analytical approach` : risk based testing (testing directed to areas of greatest risk) and requirement testing.
  - `Model based approach` : testing using statisitical information about failure rates.
  - `Methodical approaches` : failure, experience check list and quality characteristic based testing.
  - `Process/standard compliant approaches` : specified by industry standards or various agile methodologies.
  - `Dynamic and heuristic approaches` : exploratory testing, execution and evaluation are concurrent tasks.
  - `Consultative approaches` : test coverage is evaluated by domain expers outside the test team.
  - `Regression-averse approaches` : include reuse of existing test material , extensive automation of functional regression tests.
- Factors affecting test approaches and strategies include `Risks` , `Skills` , `Objectives` , `Regulations` , `Product` , and `Business` .

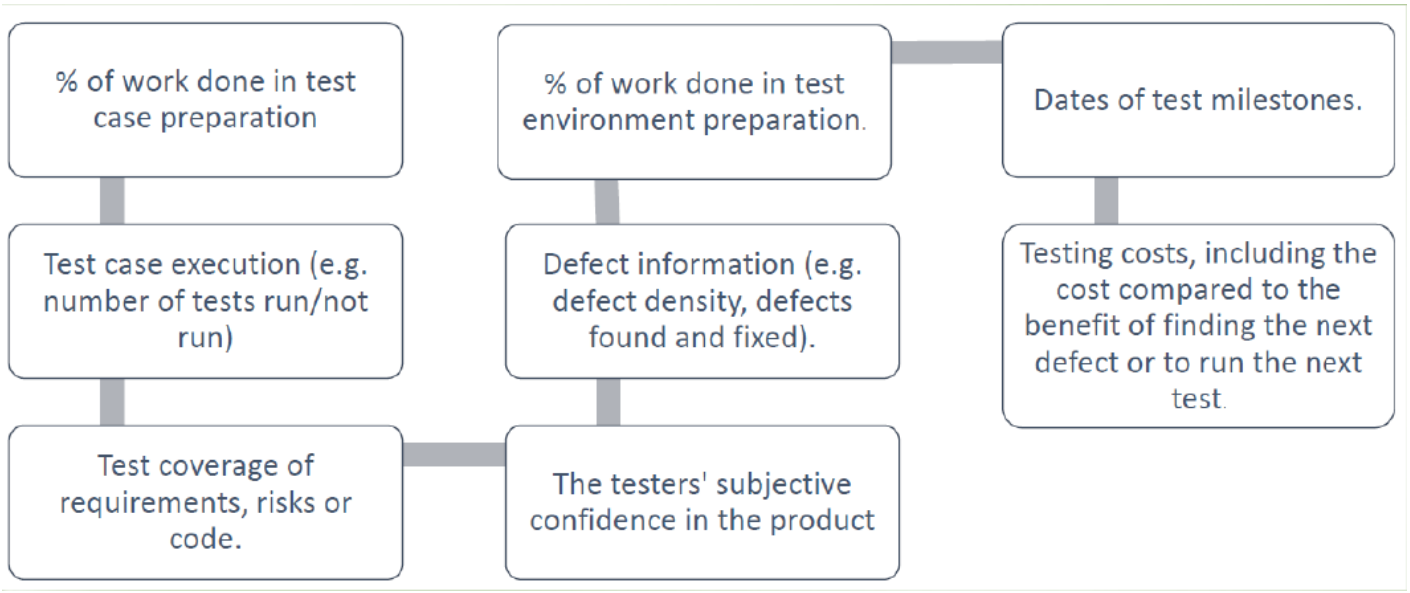
Test plan and Test strategy

- **Test Plan**: reflects test focus and project scope are defined.
  - It deals with test coverage, scheduling, features to be tested, features not to be tested, estimation and resource management.
  - Serves as a blueprint to conduct software testing activities as a defined process which is meticulously monitored and controlled by the test manager.
  - Most organization are required to adhere IEEE Standard 829-2008 Test plan documentation format.
  - Importance:
    - Test Plan helps us determine the effort needed to validate the quality of the application under test (AUT).
    - Test Plan guides our thinking.
    - Test plan contains important aspects like test estimation, test scope , test strategies so it can be reviewed by Management Team and re-used for other projects.
  - Objectives:
    - To create a set of testing tasks
    - Assign resources to each testing task
    - Estimate completion time for each testing task
    - Document testing standards
  - Characteristics of a good test plan:
    - Developed and Reviewed early
    - Clear, Complete and Specific
    - Specifies tangible deliverables that can be inspected
    - Staff knows what to expect and when to expect it
    - Realistic quality levels for goals
    - Includes time for planning
    - Can be monitored and updated
    - Includes user responsibilities
    - Based on past experience
    - Recognizes learning curves
  - How to write a Test plan?:
    - Analyze the product:
      - Review product and product documentation
      - Perform product walkthrough
      - Interview client, designer and developer

        - Design test strategy:
            - Define Scope of Testing
            - Identify Testing type
            - Document Risks and Issues
            - Create Test Logistics (Who and When)
        - Define test objectives: overall goal and achievement of the test execution
        - Define test criteria:
            - `Suspension criteria` : specify the critical suspension criteria for a test.
            - `Exit crireria` : specify the criteria that denote a successful completion of a test phase.
        - Resource planning
        - Plan test environment
        - Schedule and estimation
        - Determine test deliverables: list of all the documents, tools and other components that has been developed and maintained in support of the testing effort.
- **Test Strategy**: is a guideline to be followed to achieve the test objective and execution of test types mentioned in the testing plan.
    - It deals with test objective, test environment, test approach, automation tools and strategy, contingency plan, and risk analysis.

Test progress monitoring

- Giving feedback and visibility about test activities.
- Can be done manually or automatically



Test log template

- Test log identifier
- Description
- Activity and event entries

Test reporting

- What happened during a period of testing
- Analyzed metrics to support decisions about future actions.
- Metrics are collected at the end of a test level in order to assess:
  - The adequacy of the test objectives for that test level
  - The adequacy of the test approaches with respect to its objectives
  - The effectiveness of the testing with respect to its objectives
- Template:
  - Test summary report identifier
  - Summary
  - Variances
  - Compherensive assessment
  - Summary of results
  - Evaluation
  - Summary of activities
  - Approvals

Test control

- Any guiding or corrective actions taken as a result of information and metrics gathered and reported.
- Examples of test control actions are:
  - Making decisions based on information from test monitoring
  - Re-prioritize tests when an identified risk occurs
  - Change the test schedule due to availability of a test environment
  - Set an entry criteria requiring fixes to have been tested (confirmation tested) by a developer before accepting them into a build

Configuration management

- Establishing and maintaining the integrity of the software and related products through the project and product life cycle.
- Ensures that all items are `identified`, `version controlled`, and `tracked for changes`.
- Template:
  - Transmittal report identifier

- Transmitted items
- Location
- Status
- Approvals

Risk and testing

- Risk is the possibility of a negative or undesirable outcome, the possible problems that might endanger the objectives of the project stakeholders.
- The levels of risk is determined by the likelihood of an adverse event happening and the impact.
- For any risks you have four possibilities
  - Mitigate
  - Contingency
  - Transfer
  - Ignore
- **Project risks**:
  - the risks that surround the project's capability to deliver its objectives.
  - Typical project risks
    - Logistics or product quality problems that block tests
    - Test items that won't install in the test environment
    - Excessive change to the product that invalidates test results or requires updates to test cases, expected results and environments
    - Insufficient or unrealistic test environments that yield misleading results

Organizational Factors	Technical Issues	Supplier Issues
• Skill/staff shortage • Personal/training issues • Testers communicating their needs and test results • Improper attitude toward testing	• Problems in defining the right requirements • The extent that requirements can be met given existing constraints • The quality of design, code, and tests	• Failure of a third part • Contractual issues

- **Product risks**:
  - The possibility that the system or software might fail to satisfy some reasonable customer, user, or stakeholder expectation.

- Potential failure areas in the software.
- Directly related to the test object.

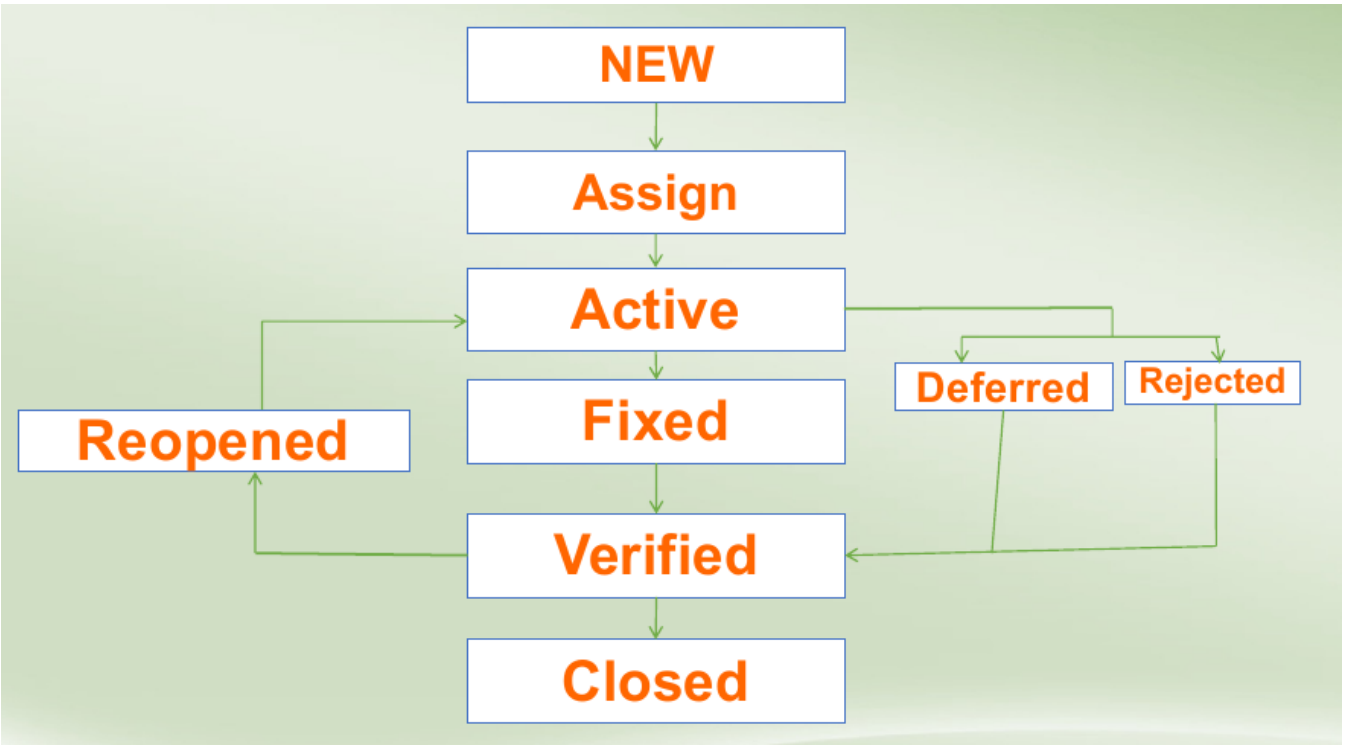
Risk analysis

- Identifying the risk items and determining the likelihood and impact for each item.
- `Risk-based testing` also involves measuring how well we are doing at finding and removing defects in critical areas.

Defect management

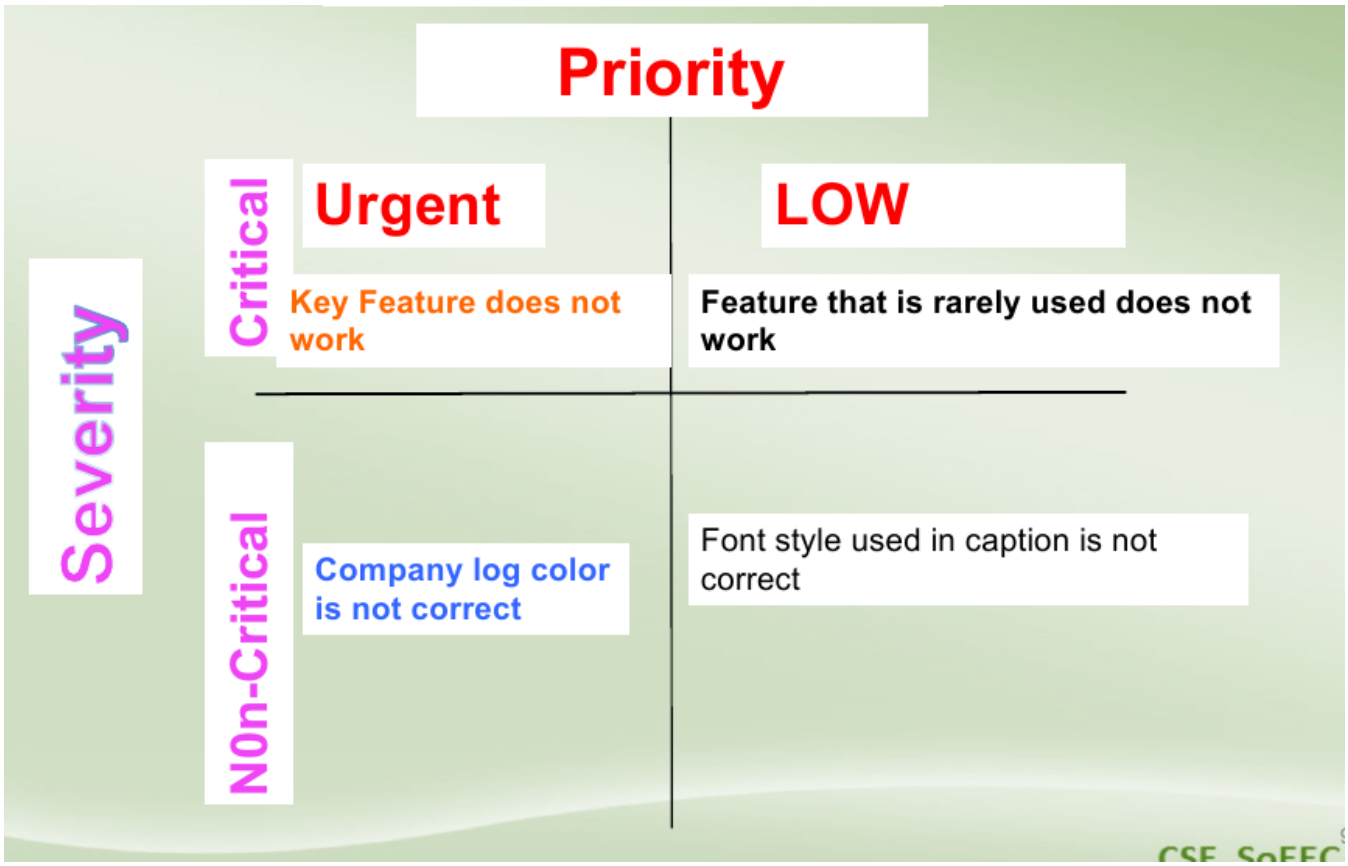
- **Defect**: discrepancies between actual and expected test outcomes.
- **Defect management**: the process of recognizing, investigating , taken action and disposing of incidents.
- **Defect report**:
  - Reporting on any efect that occured.
  - **Objectives**:
    - Provide developers and other parties 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.
  - What goes in a Defect report?:
    - A description of some situation, behavior or event that occurred.
    - One or two screens with information gathered by a defect tracking tool.
    - A description of the steps done to reproduce and isolate the incident.
    - The impact of the problem.
    - Classification information i.e. the scope , severity and priority of the defect.
- **Defect life cycle**:
  - Also known as Bug life cycle, is the journey of a defect, defect goes through different states during it's lifetime.
  - Defect states:
    - `New` : Potential defect that is raised and yet to be validated.
    - `Assigned` : Assigned against a development team to address it but not yet resolved.
    - `Active` : The Defect is being addressed by the developer and investigation is under progress. At this stage there are two possible outcomes `Deferred` or `Rejected` .
    - `Fixed/Resolved` : Defect has fixed by dev team. Fix is ready to be verified by QA.
    - `Test` : The Defect is fixed and ready for testing.
    - `Verified` : The Defect that is retested and the test has been verified by QA.

- `Closed` : The final state of the defect that can be closed after the QA retesting or can be closed if the defect is duplicate or considered as NOT a defect.
- `Reopened` : When the defect is NOT fixed, QA reopens/reactivates the defect.
- `Deferred` : When a defect cannot be addressed in that particular cycle it is deferred to future release.
- `Rejected` : A defect can be rejected for any of the 3 reasons; viz - duplicate defect, NOT a Defect, Non Reproducible.

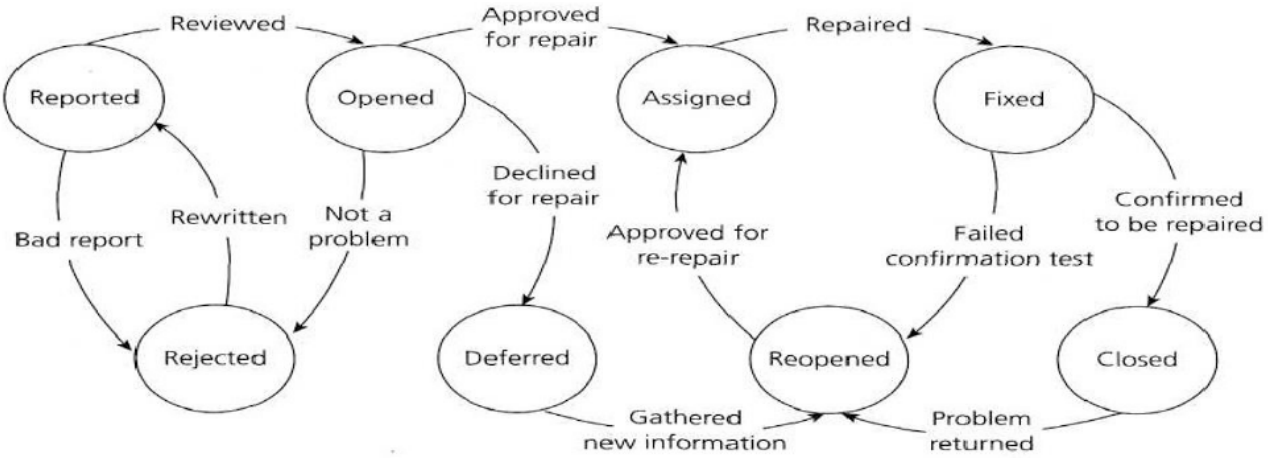


- **Defect priority**:
  - How urgently the issue can be fixed.
  - Product manager decides the Priority to fix a bug.
  - The Priority status is set by the tester to the developer mentioning the time frame to fix a defect. If Higher priority is mentioned then the developer has to fix it at the earliest.
- **Defect severity**:
  - How severe it's affecting the functionality (How bad the bug is for the system).
  - Related to the quality standard or devotion to standard.
  - Decided by test engineer.
  - Categories:
    - `Show Stopper/Blocker` : Cannot able to test application further.
    - `Major Defect` : Major functionality not working but able to test application.
    - `Minor Defect` : Bug in Functionality but in the sub module or one under the other module.
    - `Cosmetic` : Issues in location of the object or the look and feel issue.

- **Priority and Severity**:



- **Status of Defect reports**:
  - A level of priority , assigned by the test managers.
  - The risks, costs, opportunities and benefits associated with fixing or not fixing the defect assigned by the project team or a committee.
  - The root cause , captured by the programmer.
  - Conclusions and recommendations captured by the managers, programmers or others.



Chapter 4: Testing tools

Challenges in Manual testing

- Time consuming
- Tedious
- Inaccurate
- Erroneous

What's automation Testing?

Criteria for selection of Test tools

- Meeting requirements
- Technology expectations
- Training/skills
- Management aspects

Purposes of Testing tool

- Improve the efficiency
- Automate activities
- Increase relaibility

Benefits and Risks of using Testign tool

- Benefits:
    - Repetitive work is reduced
    - Greater consistency and repeatability
    - Objective assessment
    - Ease of access to information about tests or testing
    - Quick and accurate reports can be generated
- Risks:
    - Unrealistic expectations for the tool
    - Underestimating the time, cost and effort for the initial introduction of a tool.
    - Underestimating the time and effort needed to achieve significant and continuing benefits from the tool
    - Underestimating the effort required to maintain the test assets generated by the tool
    - Neglecting version control of test assets within the tool
    - Neglecting relationships and interoperability issues between critical tools
    - Risk of tool vendor going out of business, retiring the tool, or selling the tool to a different vendor

    - Poor response from vendor for support, upgrades, and defect fixes
    - Risk of suspension of open-source/free tool project
    - Inability to support a new platform

Types of Test tool

- Test management tools
- Requirements management tools
- Incident management tools
- Configuration management tools
- Functional testing tools
- Performance testing tools
- Static test tools
- Dynamic test tools
- Dev Ops tools-Continuous integration and Continuous deployment tools
- Monitoring and Monitoring tools
- Collaboration tools

Test management tools

- Test management: set of activities which covers the whole testing phase.
- Planning:
    - Risk analysis
    - Test estimation
    - Test planning
    - Test organization
- Execution:
    - Test Monitoring and control
    - Issue Management
    - Test Report and Evaluation
- Tools:
    - Google Drive
    - Jira

Test tools for static and dynamic testing

- Static testing tools:
    - Categories:
        - Flow analyzers
        - Path Tests

        - Coverage Analyzers
        - Interface Analyzers
- Dynamic testing tools:
    - Test driver
    - Test beds
    - Emulators
    - Mutation analyzers

Test design and specification tools

Test Automation Frameworks

- Linear Automation
- Behavioural Driven Development
- Modular driven
- Keyboard-Driven (Table-driven)
- Data driven
- Hybrid testing

Test Automation Flow Chart

Test Automation Process

- Define the scope of automation
- Select a testing tool
- Planning, designing, and development
- Executing test cases and building reports
- Maintaining previous test reports

Test Automation Tools

- Selenium
- Appium
- UFT
- TestComplete
- RFT
- Soap UI

Selenium

- An open-source tool that automates web browsers.
- It provides a single interface that lets you write test scripts in multiple programming language.

- **Features**:
  - Open-source
  - Language and framework support
  - Multi browser support
  - Ease of implementation
  - Less hardware usage
  - Flexibility
  - Various operating systems
  - Parallel test execution
- **Advantages**:
  - Open-source, freeware and portable tool
  - Multiple programming langage, operating system and low end hardware support
- **Disadvantages**:
  - Supports testing only web based application
  - Needs very much expertise resources
  - Does not provide any built in IDE for script generation
- **Supported tests**:
  - Functional testing
  - Regression testing
  - Acceptance testing
- **Components**:
  - `Selenium IDE`
  - `Selenium WebDriver`
  - `Selenium Grid`
- **Tools integrated with selenium**:
  - `Maven` : build automation tool.
  - `TestNG` : Java testing framework.
  - `Jenkins` : an open-source continous integration tool.
  - `Git` : version control system.

Test tool limitations

- Planning a comprehensive testing task should start as soon as possible in the initial stages of software development such as Resource planning.
- The role and purpose of testing must be defined as well as all the necessary resources, including staff for task execution, estimated time, facilities, and tools.
- The associated specifications are to be documented in the test plan. An organizational structure including test management needs to be in place and must be adapted, if necessary.
- Regular control is necessary to see if planning and project progress are in line. This may result in the need for updates and adjustments to plans to keep the test process under control.

- The basis for controlling the test process is either staff reporting or relevant data and its evaluation by appropriate tools.

Potential risks of using tools

- Unrealistic expectations
- Underestimating time, cost and effort for the introduction of a tool.
- Underestimating the time and effort needed to achieve significant and continuing benefits from the tool.
- Underestimating the effort required to maintain the test assets generated by the tool.
- Over-reliance on the tool

Chapter 5: Legal, Ethical, and Professional Aspects of Testing

Definition

- **Ethics**:
  - Branch of philosophy that deals with moral questions such as what is right or wrong, and how a person should behave in a given situation in a complex world.
  - Views:
    - Cultural relativism: argues that the particular society determines what is right or wrong based upon its cultural values.
    - Deontological ethics: argues that there are moral laws to guide people in deciding what is right or wrong.
    - Utilitarianism: rgues that an action is right if its overall affect is to produce more happiness than unhappiness in society.
- **Professional ethics**: a code of conduct that governs how members of a profession deal with each other and with third parties. Expresses ideals of human behaviour, and it defines the fundamental principles of the organization and is an indication of its professionalism.
- **Business ethics**: refers to the core principles and values of the organization and apply throughout the organization. Guide individual employees in carrying out their roles and ethical issues include the rights and duties between a company and its employees, customers, and suppliers.
- **Computer ethics**: a set of principles that guide the behaviour of individuals when using computer resources.
  - The Computer Ethics Institute (CEI) Ten commandments on computer ethics:

    i. Not use a computer to harm other people
    ii. Not interfere with other people's computer work.
    iii. Not snoop around in other people's computer files

iv. Not use a computer to steal
v. Not use a computer to bear false witness
vi. Not copy or use proprietary software for which you have not paid
vii. Not use other people's computer resources without authorization or proper compensation.
viii. Not appropriate other people's intellectual output
ix. Think about the social consequences of the program you are writing or the system you are designing
x. Always use a computer in ways that ensure consideration and respect for your fellow humans

The Ethical Software Tester

- Certified software testers shall act consistently in the public interest
- They shall act in the best interests of their client and employer
- Certified software testers shall ensure that their deliverables meet the highest professional standards
- They shall maintain independence and integrity in professional judgments
- Certified software test managers and leaders shall promote an ethical approach to the management of software testing
- They shall advance the integrity and reputation of the profession
- Certified software testers shall be supportive of their colleagues and promote cooperation with software developers
- They shall participate in lifelong learning regarding the practice of their profession and promote an ethical approach to the practice of their profession.

Professional Responsibility of Software Tester

1. Honesty and fairness in dealings with Clients
2. Responsibility for actions
3. Continuous learning to ensure appropriate knowledge to serve the client effectively

ACM Code of Professional Conduct and Ethics

- Contribute to society and human well being
- Avoid harm to others
- Be honest and trustworthy
- Be fair and act not to discriminate
- Respect property rights
- Respect intellectual property
- Respect the privacy of others
- Respect confidentiality

Chapter 6: Software Quality Assurance

What's Quality?

- The degree to which a system, component, or process meets specified requirements. (IEEE)
- Quality is conformance to specifications. (British Defense Industries Quality Assurance Panel)
- The degree to which a system, component, or process meets customer or user needs or expectations. (IEEE)
- Quality is conformance to requirements. (Philip Crosby)
- Quality is synonymous with customer needs and expectations. (RJ Mortiboys)
- The totality of features and characteristics of a product or a service that bear on its ability to satisfy stated or implied needs. (ISO 8402)
- Quality is meeting the (stated) requirements of the customer- now and in the future. (Mike Robinson)

Major Quality Characteristics

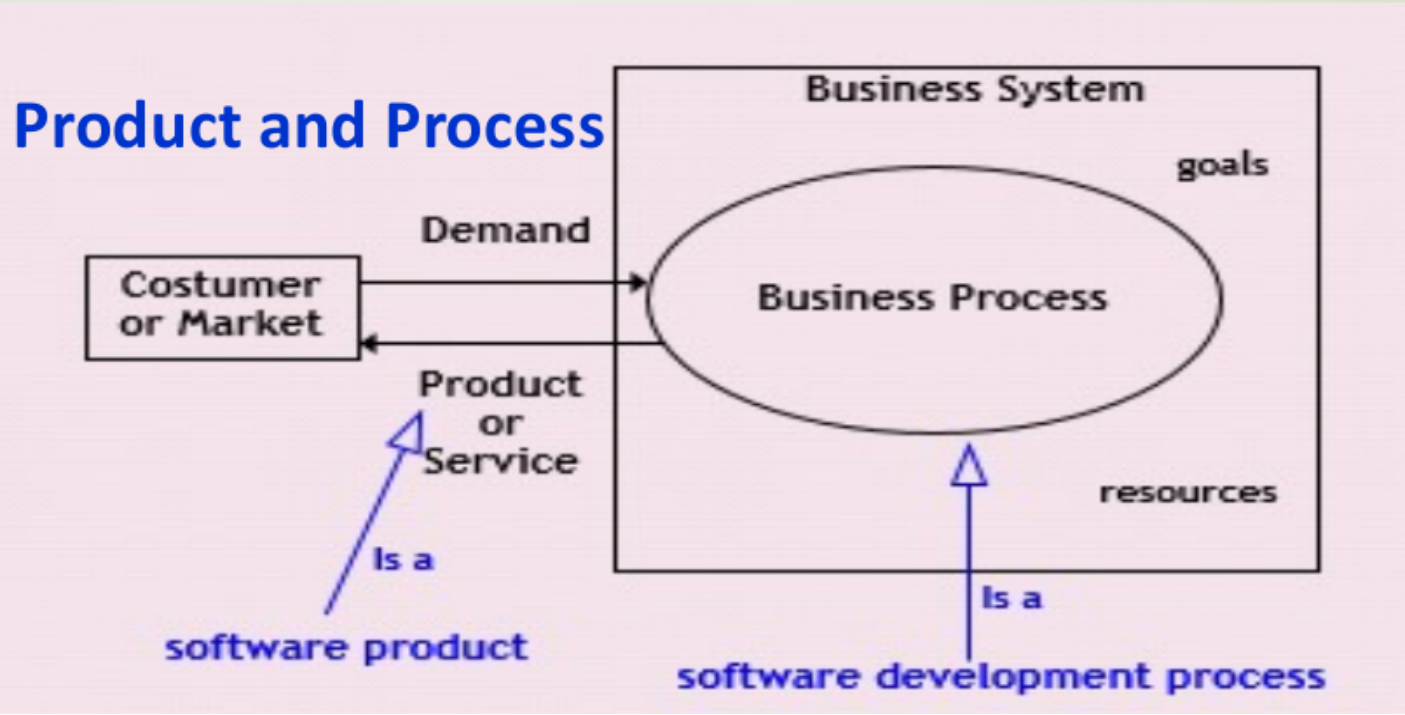
1. Functionality
2. Reliability
3. Usability
4. Efficiency
5. Maintainability
6. Portability

Problems with Software Quality

- Software specifications are usually incomplete and often inconsistent.
- Some GAPs between customer quality requirements (efficiency, reliability) and developer quality requirements (maintainability, reusability).
- Some quality requirements are hard to specify in an unambiguous way:
    - directly measurable qualities (e.g., errors/KLOC),
    - indirectly measurable qualities (e.g., usability).

Views of Software Quality

- **Transcendental view**: Quality is something that can be recognized through experience but is not defined in some tractable form.
- **User view**: the extent to which a product meets user needs and expectations. User is concerned with whether or not the product is fit for use.
- **Manufacturing view**: quality is seen as conforming to requirements. Any deviation from the stated requirements is seen as reducing the quality of the product.
- **Product view**: if a product is manufactured with good internal characteristics then it will have good external quality.
- **Value-Based views**: Quality is a measure of excellence, and value is a measure of worth. The central view in this case is how much a customer is willing to pay for a certain level of quality.



Process and product quality

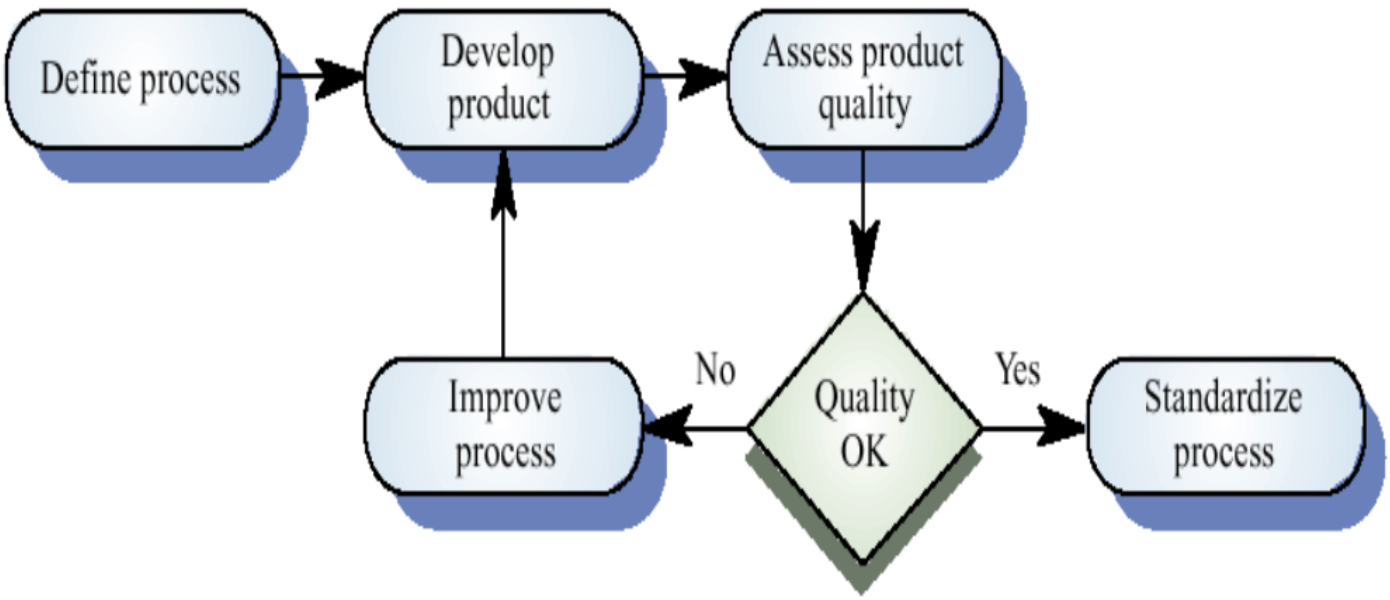
- Software product:
  - computer programs (sources and executables) + associated documentation
  - Custom product: developed for a particular customer, according to its specifications.
  - Generic product: developed for a general market, to be sold to a range of different customers.
  - Types of software products:
    - Business support software
    - Personal productivity software
    - Embedded software
  - Software development process:
    - a set of activities whose goal is the development or evolution of a software product.
    - Generic activities:
      - Specification - what the system should do and its development constraints.
      - Development - production of the software system.
      - Validation - checking that the software is what the customer wants.
      - Evolution - changing the software in response to changing demands.
  - Product quality: software product should deliver the required functionality (functional requirements) with the required quality attributes (non–functional requirements).
  - Principal product quality factors:
    - Process quality: a good process is usually required to produce a good product.
    - People quality: the capabilities of the developers is the main determinant.
    - Development technology
    - Budget and schedule
  - Attributes:
    - Efficiency

        - Usability,
        - Dependability
            - reliability,
            - availability,
            - security,
            - safety
        - Maintainability
        - Resilience
        - Robustness
        - Understandability
        - Testability
        - Adaptability
        - Modularity
        - Simplicity
        - Portability
        - Reusability
        - Learnability
    - Attributes of good software:
        - Efficiency
        - Usability
        - Dependability
        - Maintainability

Attributes of good software beyond delivering the required functionality

- Efficiency
- Usability
- Dependability (Reliability, avaliability, security and safety)
- Maintainability

Process based Quality



Process quality attributes

- Understandability
- Visibility
- Supportability
- Acceptability
- Reliability
- Robustness
- Maintainability
- Rapidity

Measuring Quality

- Allows us to have a quantitative view of the quality concept and establish baslines for qualities.
- Enables organizations to know how much improvement in quality is achieved at a certain cost incurred due to process improvement.
- **Measurement of User's View**:
  - Encompasses several quality factors, such as functionality, reliability and usability.
  - Can be measured by designing at least one test case for each functionality.
- **Measurement of Manufacturer's View**:
  - Defect count: *How many defects have been detected?*
  - Rework cost: *How much does it cost to fix the known defects?*
    - Development Rework Cost:
      - Cost incurred before a product is released to customers.
      - It's a measure of development efficiency.
      - Low development rework cost = High development efficiency
    - Operation Rework Cost:
      - Cost incurred when a product is in operation.
      - Measure of the delivered quality of the product in operation.
      - Low operation rework cost = High delivered quality of the product in operation (customers have not encountered any defect).

Quality factors and criteria

- **Quaity factors**:
  - Represents a behavioural characteristic and external attributes of a system.
  - Quality factors stated by actors of the system:
    - Customers: may want `efficient` and `reliable` software with less concern for portability.
    - Developers: strive to meet customer needs, at the same time making the product `portable` and `reusable` to reduce the cost of development.
    - Quality Assurance Team: interested in the `testability` of the system so that some other factors can be easily verified through testing.

- **McCall's Quality Factors**:
  - Contains 11 quality factors that can be groped into 3 borad categoris, *Product operation*, *Product revision* and *Product transition*.

Quality Categories	Quality Factors	Broad Objectives
Product operation	Correctness	Does it do what the customer wants?
	Reliability	Does it do it accurately all of the time?
	Efficiency	Does it quickly solve the intended problem?
	Integrity	Is it secure?
	Usability	Can I run it?
Product revision	Maintainability	Can it be fixed?
	Testability	Can it be tested?
	Flexibility	Can it be changed?
Product transition	Portability	Can it be used on another machine?
	Reusability	Can parts of it be reused?
	Interoperability	Can it interface with another system?

- **Quality Critera**
  - An attribute of a quality factor that is related to software development.

Quality Factor	Quality Criteria
Correctness	Traceability, Completeness, and Consistency
Reliability	Consistency, Accuracy, Error tolerance, and Simplicity
Efficiency	Execution and Storage efficiency
Integrity	Access control and Access audit
Usability	Operability, Training, and Communicativeness
Maintainability	Consistency, Simplicity, Conciseness, Self-descriptiveness, and Modularity
Testability	Simplicity, Instrumentation, Self-descriptiveness, and Modularity
Flexibility	Self-descriptiveness, Expandability, Generality and Modularity
Portability	Self-descriptiveness, Modularity, Software-system independence,and Machine independence
Reusability	Self-descriptiveness, Modularity, Generality, Software-system independence,and Machine independence
Interoperability	Modularity, Communications commonality, and Data commonality

Quality Metrics

- Measure that captures some aspect of a quality criterion.
- Steps:
    i. Formulate a set of relevant questions concerning the quality criteria and seek a “yes” or “no” answer for each question.
    ii. Divide the number of “yes” answers by the number of questions to obtain a value in the range of 0 to 1. The resulting number represents the intended quality metric.

Quality Characteristics

- **ISO 9126 Quality Characteristics**:
    - Standardized software quality document.

Category	Sub-category	Description
Functionality	Sustainability, Accuracy, and Interoperability	A set of attributes that bear on the existence of a set of functions and their specified properties.
Reliability	Maturity, Fault tolerance, and Recoverability	A set of attributes that bear on the capability of software to maintain its performance level under stated conditions for a stated period of time.
Usability	Understandability, Learnability and Operability	A set of attributes that bear on the effort needed for use and on the individual assessment of such use by a stated or implied set of users.
Efficiency	Time and Resource behavior	A set of attributes that bear on the relationship between the software’s performance and the amount of resource used under stated conditions.
Maintainability	Analyability, Changeability, Stability, and Testability	A set of attributes that bear on the effort needed to make specified modifications.
Portability	Adaptability, Installability, Conformance, and Replaceability	A set of attributes that bear on the ability of software to be transferred from one environment to another.

Difference between McCall and ISO 9126 models

- The ISO 9126 model emphasizes characteristics visible to the users, whereas the McCall model considers internal qualities.

- In McCall's model, one quality criterion can impact several quality factors, whereas in the ISO 9126 model, one sub-characteristic impacts exactly one quality characteristic.
- A high-level quality factor, such as testability, in the McCall model is a low-level sub-characteristic of maintainability in the ISO 9126 model.

Concerns with the quality models

- There is no consensus about what high-level quality factors are more important at the top level.
- There is no consensus regarding what is a top-level quality factor/ characteristic and what is a more concrete quality criterion/subcharacteristic.

Software Quality Standards

- **ISO 9000:2000 standard**:
  - International Software Quality Standard that based on the following eight principles:
    - Customer Focus
    - Leadership
    - Involvement of People
    - Process Approach
    - System Approach to Management
    - Continual Improvement
    - Factual Approach to Decision Making
    - Mutually Beneficial Supplier Relationship

Software Quality Assurance Group

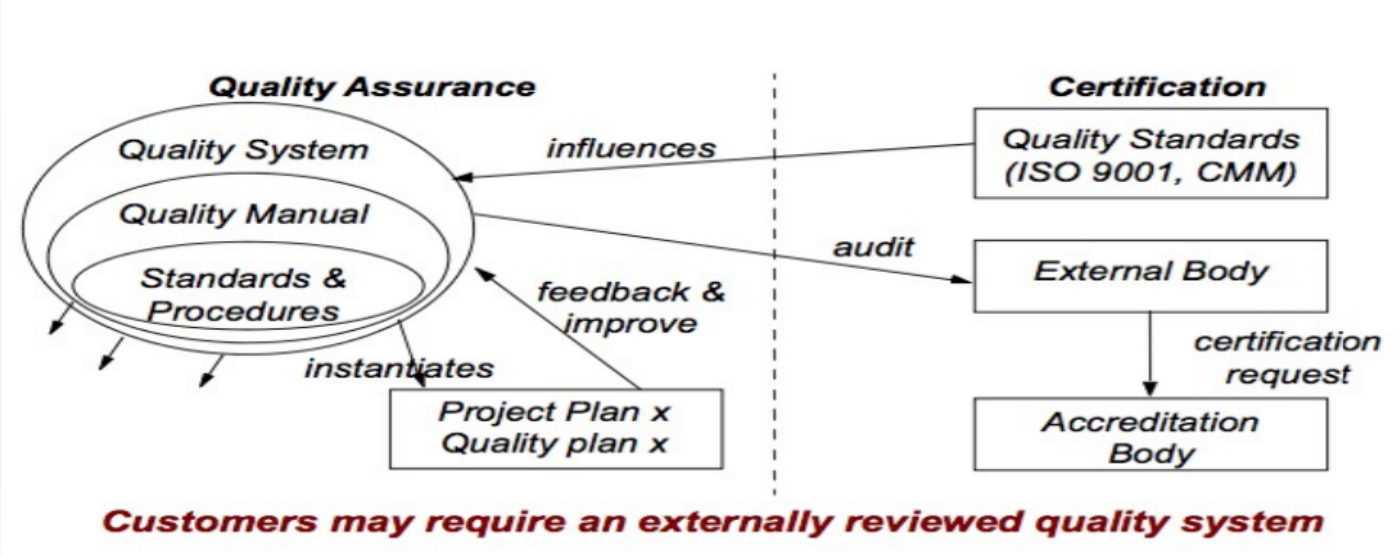
- Team of people with the neccessary training and skills to ensure that:
  - All necessary actions are taken during the development process.
  - So that the resulting software conforms to established technical requirements.
- Must look at software from customer's perspective assessing its technical merits.

- **Activities**:
  - Prepare SQA plan for the project.
  - Participate in the development of the project's software process Description.
  - Review software engineering activities to verify compliance with the defined software process.
  - Audit designated software work products to verify compliance with those defined as part of the software process.
  - Ensure that any deviations in software or work products are documented and handled according to a documented procedure.
  - Record any evidence of noncompliance and reports them to management.

IEEE Standard for SQA Plan

- The IEEE Standards Association has developed a standard (Std 730-1989) for software quality assurance plans.
- Document structure:
  - Purpose
  - Reference Documents
  - Management:
    - Organization
    - Tasks
    - Responsibilities
  - Documentation:
    - Purpose
    - Minimum documents:
      - SRS (Software Requirement Specification)
      - SDD (Software Design Description)
      - SVVP (Software Verification and Validation Plan)
      - SVVR (Software Verification and Validation Report)
      - User documentation & SCMP (Software Configuration Management Plan)
  - Standards, Practices, Conventions, and Metrics
  - Reviews and Audits
  - Tests
  - Problem Reporting
  - Tools, Techniques and Methodologies
  - Code Control
  - Media Control
  - Supplier Control
  - Records
  - Training
  - Risk Management

Quality System



ISO 9000

- ISO 9000 is an international set of standards for quality management applicable to a range of organizations from manufacturing to service industries.
- ISO 9001 must be instantiated for each organization.
- ISO 90003 interpret: ISO 9001 for the software developer.

CMM and CMMI

- **CMM**:
    - A framework that lays out five maturity levels for continual process improvement. This framework is integral to most management systems that aim to improve the quality of development and delivery for all products and services.
    - Stages:
        i. `Initial` : The software process is characterised as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

        ii. `Repeatable` : Basic project management processes are established to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

        iii. `Defined` : The software process for both management and engineering activities is documented, standardised and integrated into all processes for the organisation. All projects use an approved version of the organisation’s standard software process for developing and maintaining software.

iv. `Managed` : Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

v. `Optimising` : Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

- **CMMI**:
  - Successor of CMM which uses two representations of the maturity of the processes.
  - PCMM:
    - Strong emphasis on learning and professional development, both at the individual and organizational levels.
    - Culture that empowers individuals and promotes a participatory environment.
    - Effective professional mentoring programs.
    - Emphasis on continuous improvement.
  - Six-Sigma:
    - It is a customer based approach realizing that defects are expensive.
    - It is a structured application of tools & techniques applied on the project basis to achieve sustainable results.
    - Six Sigma processes will produce less than 3.4 defects /mistakes for a million opportunities i.e. 99.99% ,perfection [Called five 9's].