Software Testing

Life Cycle (STLC)

Contents

Part 1: Quality

Quality has two elements

- QA - Quality Assurance
- QC - Quality Control

What is Quality ?

- **Quality from the**
  - Customer’s Viewpoint Fitness for use, or other customer needs
  - Producer’s Viewpoint Meeting requirements

Quality Function

- Software quality includes activities related to both
  - Process, and the
  - Product
- Quality Assurance is about the work process
- Quality Control is about the product

What is Quality Assurance?

- Quality assurance activities are work process oriented.
- They measure the process, identify deficiencies, and suggest improvements.
- The direct results of these activities are changes to the process.
- These changes can range from better compliance with the process to entirely new processes.
- The output of quality control activities is often the input to quality assurance activities.
- Audits are an example of a QA activity which looks at whether and how the process is being followed. The end result may be suggested improvements or better compliance with the process.

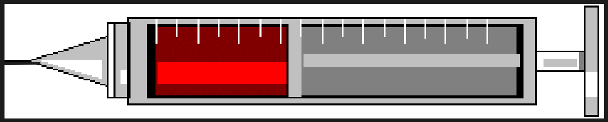
What is Quality Control?

- Quality control activities are work product oriented.
- They measure the product, identify deficiencies, and suggest improvements.
- The direct results of these activities are changes to the product.
- These can range from single-line code changes to completely reworking a product from design.
- They evaluate the product, identify weaknesses and suggest improvements.
- Testing and reviews are examples of QC activities since they usually result in changes to the product, not the process.
- QC activities are often the starting point for quality assurance (QA) activities.

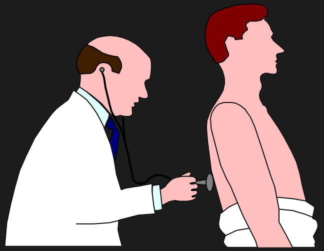
Prevention and Detection

Prevention is better than cure . . .

. . . but not everything can be prevented!



Prevention



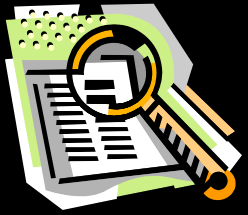
Detection

Cure

QA and QC Broad Difference

QA	QC
Assurance	Control
Process	Product
Preventive	Detective
Quality Audit	Testing

Quality... it's all about the End-User



Does this software product work as advertised?
Functionality, Performance, System & User Acceptance ... Testing

Will the users be able to do their jobs using this product?
Installability, Compatibility, Load/Stress ... Testing

Can they bet their business on this software product?
Reliability, Security, Scalability ... testing

Real World Software Testing

Software Testing Genius

Part 2: Testing

Testing

What is testing

Objectives of Testing

What is Testing?

Definition:

- Testing is process of trying to discover every conceivable fault or weakness in a work product.
- Testing is a process of executing a program with the intent of finding an error.
- A good test is one that has a high probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an as yet undiscovered error

What is Testing?

- Testing is a process used to identify the correctness, completeness and quality of developed computer software. Testing, apart from finding errors, is also used to test performance, safety, fault-tolerance or security.

- Software testing is a broad term that covers a variety of processes designed to ensure that software applications function as intended, are able to handle the volume required, and integrate correctly with other software applications.

What are the “right” things to test?

- Are you testing an entire product or some components?
- Which functions are ***critical*** for a product?
- Which functions are ***highly visible*** when they fail?
- Which functions have been ***highly error-prone***?
- Which functions are ***most often used***?
- Which functions are ***changed most often / recently***?
- Which functions suffered ***turnover of employees***?
- Where is high ***geographic distribution of work***?
- Which functions require ***new skills / technology***?

Looks Impossible to do it all!

What is objective of Testing?

- Objective of testing is to find all possible bugs (defects) in a work product

- Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should.

Setting up criteria for testing

Criteria can be at phase level

- **Entry Criteria**
  - Parallelism Vs Ready for use
- **Exit Criteria**
  - Completeness Vs Risk of release
- **Suspension Criteria**
  - Show stopper bugs
  - Crossing a threshold number of bugs
  - Developers producing a new version making the old one redundant
- **Resumption Criteria**
  - Above hurdles being cleared

General Testing Principles

- Testing shows presence of Defect
- Exhaustive Testing is Impossible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context dependent
- Absence-of-error fallacy

Summary

Identify defects

- when the software doesn't work

Verify that it satisfies specified requirements

- verify that the software works

Mature view of software testing

- A mature view of software testing is to see it as a process of reducing the risk of software failure in the field to an acceptable level [Bezier 90].

What exactly Does a Software Tester Do?

- The Goal of a software tester is to find defects
- And find them as early as possible.

What does testing mean to testers?

Testers hunt errors

- Detected errors are celebrated - for the good of the work product

Testers are destructive - but creatively so

- Testing is a positive and creative effort of destruction

Testers pursue errors, not people

- Errors are in the work product, not in the person who made the mistake

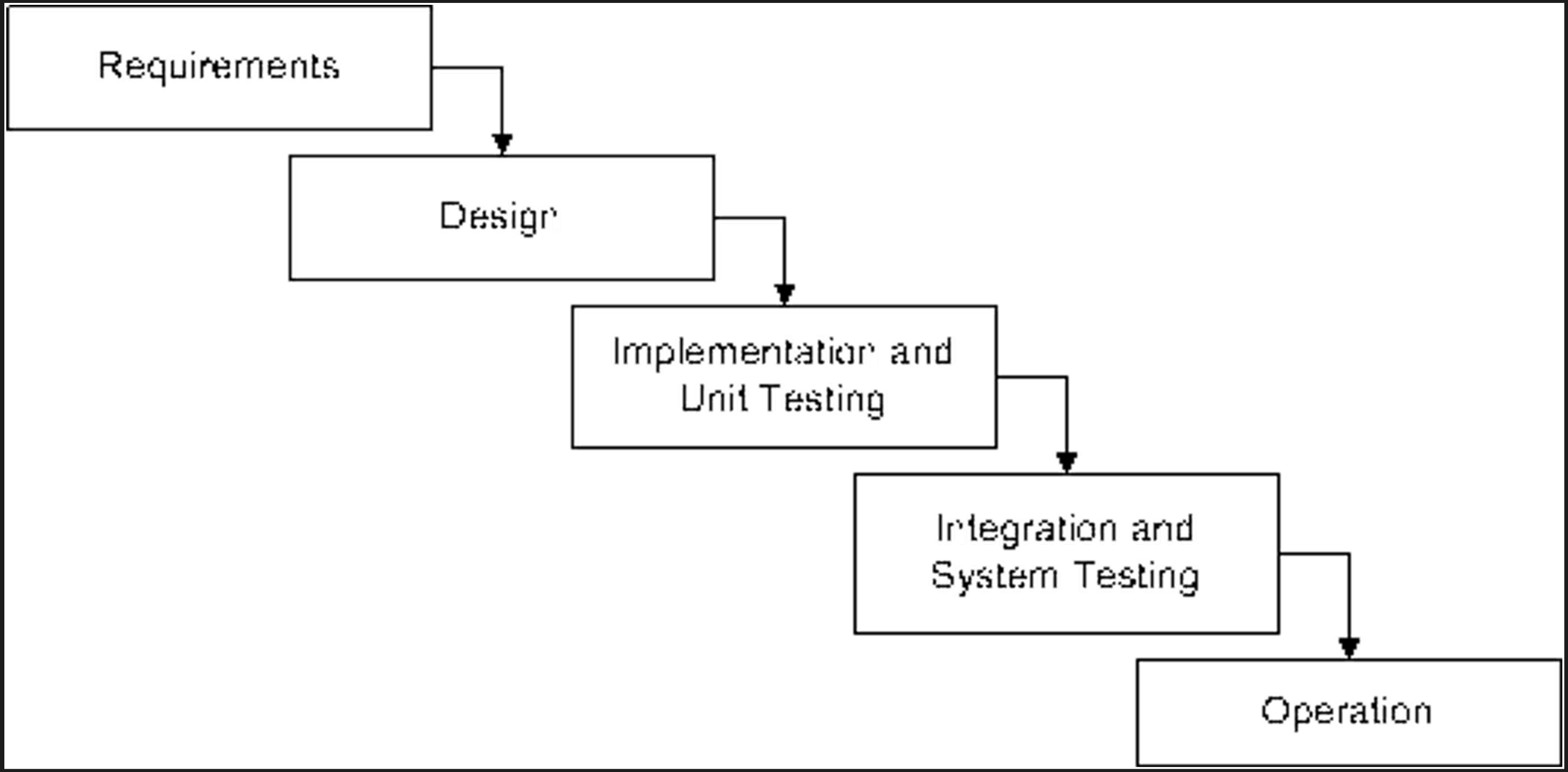
Testers add value

- by discovering errors as early as possible

How testers do it?

- By examining the user’s requirements, internal structure and design, functional user interface etc
- By executing the code, application software executable etc

Waterfall Model for Testing


Requirements
Design
Implementation and Unit Testing
Integration and System Testing
Operation

Advantages of Waterfall Model

- Simple and easy to use.

- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.

- Phases are processed and completed one at a time.

- Works well for smaller projects where requirements are very well understood.

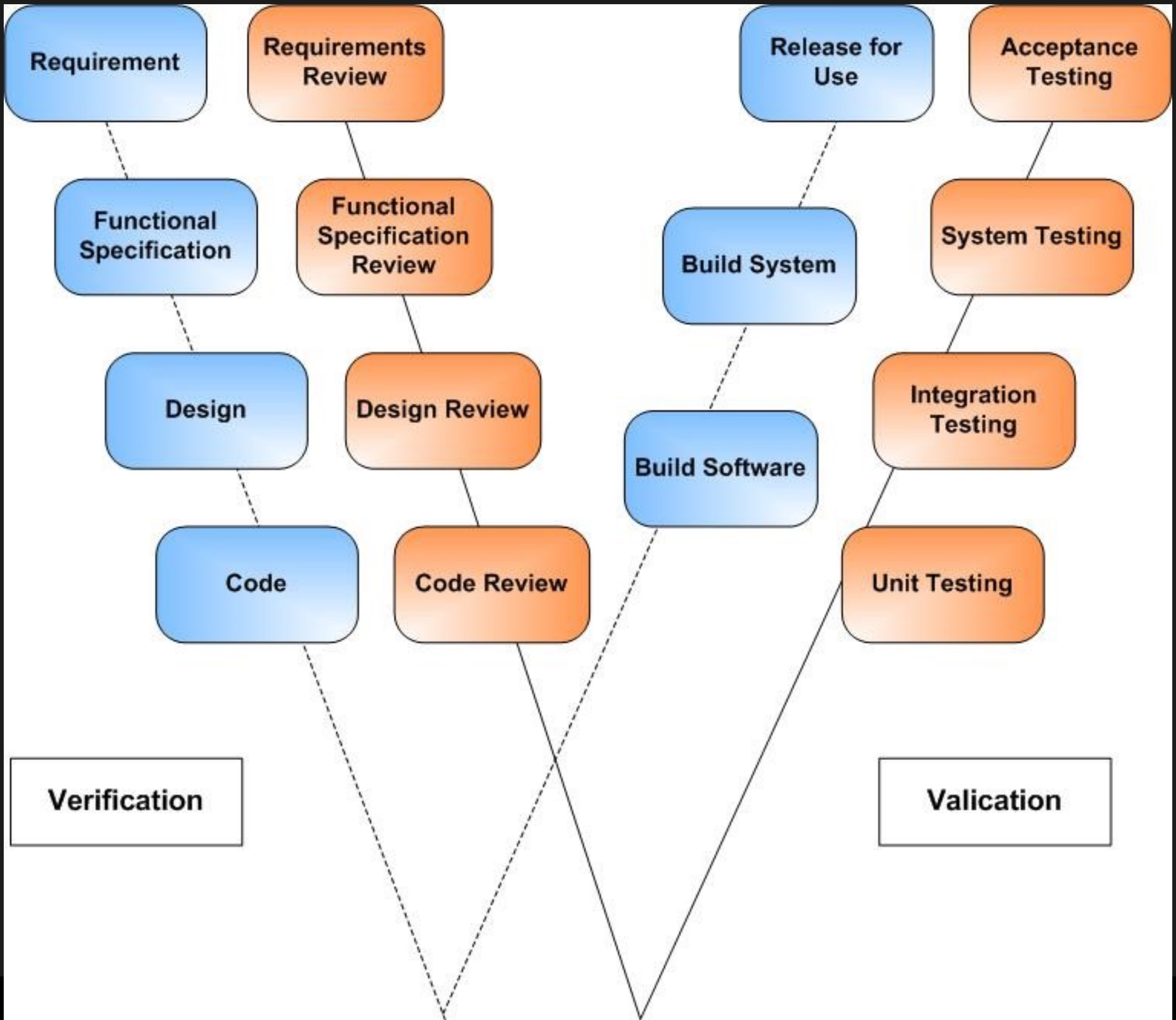
Disadvantages of Waterfall Model

- Adjusting scope during the life cycle can kill a project
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Poor model for complex and object-oriented projects.
- Poor model where requirements are at a moderate to high risk of changing.

Disadvantages of Waterfall Model

- It is a process of establishing requirements, designing, building and testing a system, done as a series of smaller development
- Increment produced by Iteration is tested at several levels as part of its development
- Regression testing is important after all iterations
- Verification and verification can be carried outs on each iteration

W-Model for Testing



Advantages of W - Model

- Simple and easy to use.
- Each phase has specific deliverables.
- Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
- Works well for small projects where requirements are easily understood.

Disadvantages of W - Model

- ❖ Very rigid, like the waterfall model.
- ❖ Little flexibility and adjusting scope is difficult and expensive.
- ❖ Software is developed during the implementation phase, so no early prototypes of the software are produced.
- ❖ Model doesn’t provide a clear path for problems found during testing phases.

Part 3: Different stages of SDLC with STLC

Stages of SDLC with STLC

Different stages of SDLC with STLC

Stage-1 Requirement Gathering

- **DA:** Defining requirements to establish specifications is the first step in the development of software.

- **TA:** Reviewing the requirements e.g the requirements should not have ambiguous words like (may or may not). It should be clear and concise.

Different stages of SDLC with STLC

Stage-2 Functional Specification

- **DA:** It describes the product's behaviors as seen by an external observer, and contains the technical information and data needed for the design. The functional specification defines what the functionality will be

- **TA:** In order to make the functional specifications accurate we have review our functional specifications.

Different stages of SDLC with STLC

Stage-3 Design

- **DA:** The software specifications are transformed in to design models that describe the details of the data structures, system architecture, interface and the components.

- **TA:** Each design product is reviewed for quality before moving the next phase of the software development.

Different stages of SDLC with STLC

Stage-4 Code

- **DA:** In this phase the designs are translated into code.
- **TA:** Code review is a process of verifying the source code. Code review is done the find and fix the defects that are overlooked in the initial development phase, to improve overall quality of code.

Different stages of SDLC with STLC

Stage-5 Building Software

- **DA:** In this phase we build different software units and integrate them one by one to build single software.
- **TA:** Unit testing & Integration testing

Different stages of SDLC with STLC

Stage-6 Building system

- **DA:** After the software has been build we have the whole system considering all the non-functional requirements like installation procedure, configuration etc.

- **TA:** System testing & Acceptance testing

Different stages of SDLC with STLC

Stage-7 Release for use

- After the whole product has been developed and the required level of quality has been achieved and the software is release for the actual use of the customers.

Part 4: Testing Techniques

Testing Techniques

- Verification and Validation

Verification

- Reviews
- Walkthrough
- Inspection

Verification “What to Look For?”

- **Find all the missing information**
  - Who
  - What
  - Where
  - When
  - Why
  - How

Peer Review

- Simply giving a document to a colleague and asking them to look at it closely which will identify defects we might never find on our own.

Walkthrough

Informal meetings, where participants come to the meeting;and the author gives the presentation.

- **Objective:**
  - To detect defects and become familiar with the material
- **Elements:**
  - A planned meeting where only the presenter must prepare
  - A team of 2-7 people, led by the author
  - Author usually the presenter.
- **Inputs:**
  - Element under examination, objectives for the walkthroughs applicable standards.
- **Output:**
  - Defect report

Inspection

Formal meeting, characterized by individual preparation by all participants prior to the meeting.

- **Objectives:**
  - To obtain defects and collect data.
  - To communicate important work product information .
- **Elements:**
  - A planned, structured meeting requiring individual preparation by all participants.
  - A team of people, led by an impartial moderator who assure that rules are being followed and review is effective.
  - Presenter is “reader” other than the author.
  - Other participants are inspectors who review,
  - Recorder to record defects identified in work product

Checklists : the verification tool

- An important tool specially in formal meetings like inspections
- They provide maximum leverage on on verification
- There are generic checklists that can be applied at a high level and maintained for each type of inspection
- There are checklists for requirements,functional design specifications, internal design specifications, for code

Validation Strategies

There are two main strategies for validating software

- White Box testing
- Black Box testing

Validation Strategies

White Box Testing

- Deals with the internal logic and structure of the code
- The tests are written based on the white box testing strategy incorporate coverage of the code written, branches, paths, statements and internal logic of the code etc.
- Normally done the developers

White Box testing Methods

White Box Testing can be done by:

- Data Coverage
- Code Coverage

White Box testing Methods

Date Coverage

- Data flow is monitored or examined through out the program. E.g. watch window we use to monitor the values of the variables and expressions.

White Box testing Methods

Code Coverage

- It’s a process of finding areas of a program not exercised by a set of test cases,
- Creating additional test cases to increase coverage
- Code coverage can be implemented using basic measure like, statement coverage, decision coverage, condition coverage and path coverage

Validation Strategies

Black Box Testing

- Does not need any knowledge of internal design or code
- Its totally based on the testing for the requirements and functionality of the work product/software application.
- Tester is needed to be thorough with the requirement specifications of the system and as a user, should know how the system should behave in response to the particular action.

Black Box testing Methods

The following are commonly used Black Box methods :

- Equivalence partitioning
- Boundary-value analysis
- Error guessing

Equivalence Partitioning

- An equivalence class is a subset of data that is representative of a larger class.

- Equivalence partitioning is a technique for testing equivalence classes rather than undertaking exhaustive testing of each value of the larger class.

Equivalence Partitioning

If we expect the same result from two tests, you consider them equivalent. A group of tests from an equivalence class if,

- They all test the same thing
- If one test catches a bug, the others probably will too
- If one test doesn’t catch a bug, the others probably won’t either

Equivalence Partitioning

For example, a program which edits credit limits within a given range ($10,000-$15,000) would have three equivalence classes:

- Less than $10,000 (invalid)
- Between $10,000 and $15,000 (valid)
- Greater than $15,000 (invalid)

Equivalence Partitioning

- Partitioning system inputs and outputs into ‘equivalence sets’
  - If input is a 5-digit integer between 10,000 and 99,999 equivalence partitions are <10,000, 10,000-99,999 and >99,999
- The aim is to minimize the number of test cases required to cover these input conditions

Equivalence Partitioning

***Equivalence classes* may be defined according to the following guidelines:**

- If an input condition specifies a *range*, one valid and two invalid equivalence classes are defined.

- If an input condition requires a specific *value*, then one valid and two invalid equivalence classes are defined.

- If an input condition is *Boolean*, then one valid and one invalid equivalence class are defined.

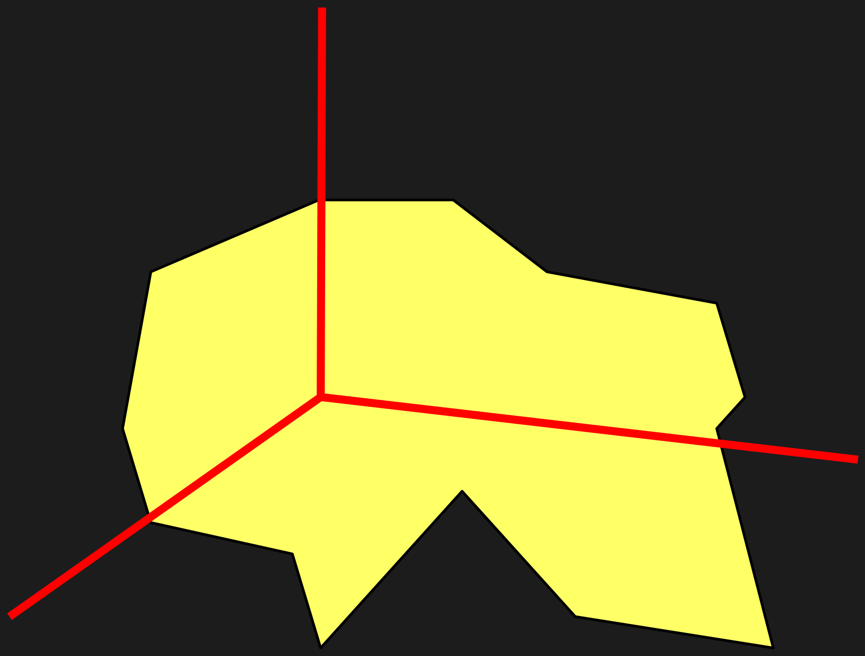
Equivalence Partitioning Summary

- Divide the input domain into classes of data for which test cases can be generated.
- Attempting to uncover classes of errors.
- Based on equivalence classes for input conditions.
- An equivalence class represents a set of valid or invalid states
- An input condition is either a specific numeric value, range of values, a set of related values, or a Boolean condition.
- Equivalence classes can be defined by:
- If an input condition specifies a range or a specific value, one valid and two invalid equivalence classes defined.
- If an input condition specifies a Boolean or a member of a set, one valid and one invalid equivalence classes defined.
- Test cases for each input domain data item developed and executed.

Boundary value analysis

- “Bugs lurk in corners and congregate at boundaries…”

Boris Beizer



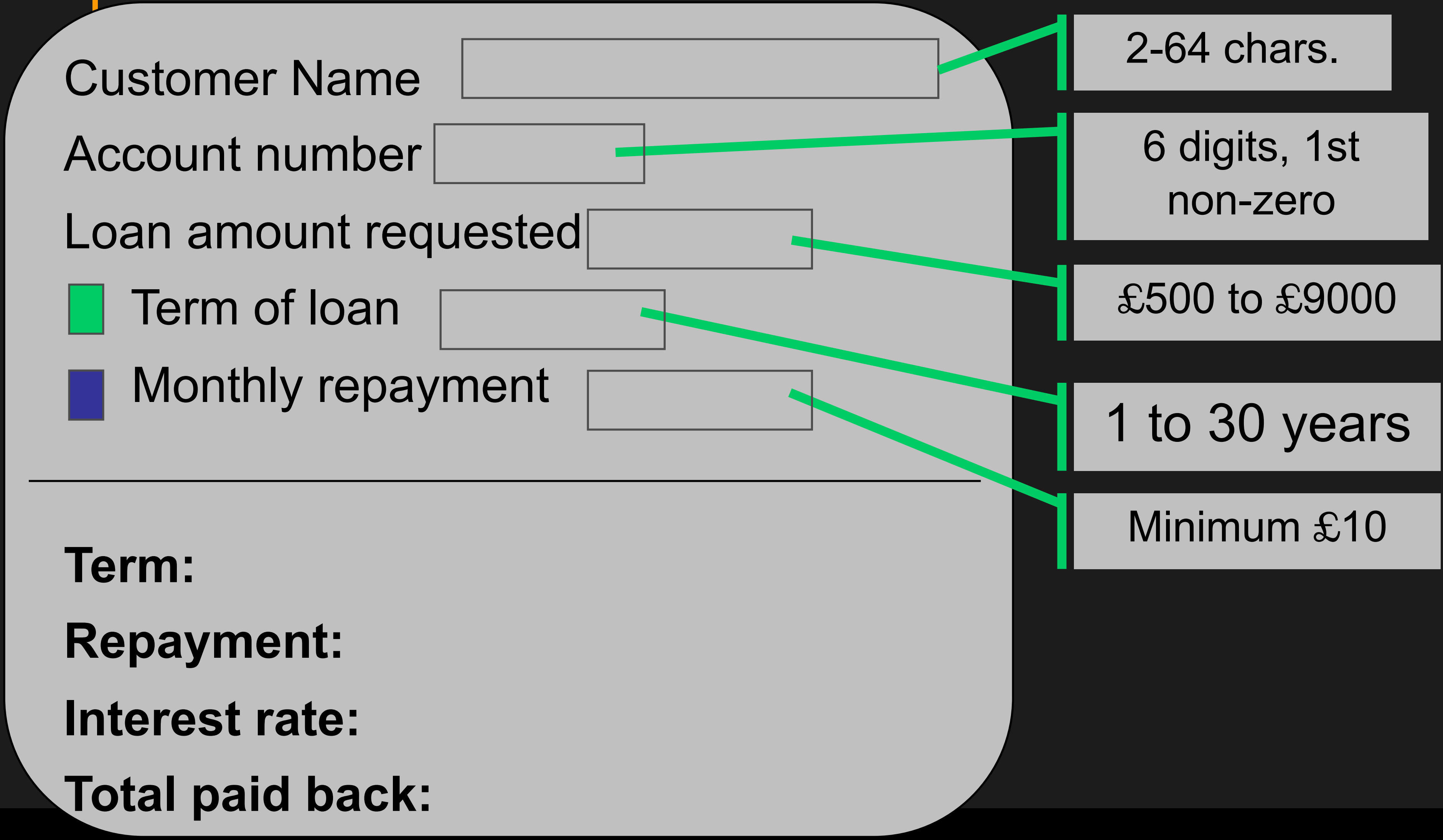
Boundary value analysis

- A technique that consists of developing test cases and data that focus on the input and output boundaries of a given function.
- In same credit limit example, boundary analysis would test:
    - Low boundary plus or minus one ($9,999 and $10,001)
    - On the boundary ($10,000 and $15,000)
    - Upper boundary plus or minus one ($14,999 and $15,001)

Boundary value analysis

- Large number of errors tend to occur at boundaries of the input domain
- BVA leads to selection of test cases that exercise boundary values
- BVA complements equivalence partitioning. Rather than select any element in an equivalence class, select those at the "edge' of the class
- Examples:
- For a range of values bounded by a and b, test (a-1), a, (a+1), (b-1), b, (b+1)
- If input conditions specify a number of values n, test with (n-1), n and (n+1) input values
- Apply 1 and 2 to output conditions (e.g., generate table of minimum and maximum size)

Example: Loan application



Customer Name

Account number

Loan amount requested

🟩 Term of loan

🟦 Monthly repayment

Term:

Repayment:

Interest rate:

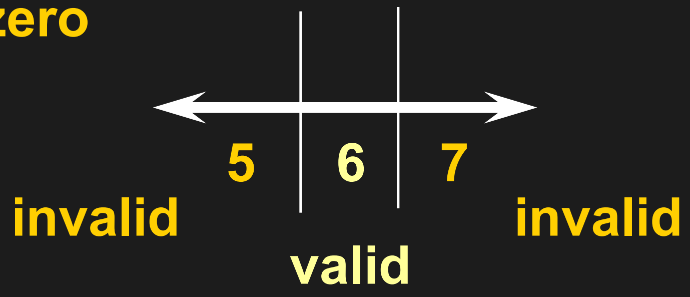
Total paid back:

Account number

First character: valid: non-zero

invalid: zero

Number of digits:



Conditions	Valid Partitions	Invalid Partitions	Valid Boundaries	Invalid Boundaries
Account number	6 digits 1st non-zero	< 6 digits ~~> 6 digits~~ ~~1st digit = 0~~ non-digit	100000 999999	5 digits ~~7 digits~~ 0 digits

Error Guessing

- Based on the theory that test cases can be developed based upon the intuition and experience of the Test Engineer

- For example, in an example where one of the inputs is the date, a test engineer might try February 29,2000 or 9/9/99

Part 5: Various Types of Testing

Validation Activities

Validation is done at two levels

- **Low Level**
  - Unit testing
  - Integration Testing
- **High Level**
  - Function Testing
  - System Testing
  - Acceptance Testing

Unit Testing

- Searches for defect and verifies the functionality of software, depending upon the context of the development
- It includes testing of functional and non-functional characteristics
- It occurs with access to code being tested and with the support of development environment
- Defects are fixed as soon as they are found with out formally recording incident
- If test cases are prepared and automated before coding, it is termed as test-first approach or test-driven development.

Integration Testing

- Integration testing tests interface between components, interaction to different parts of system.
- Greater the scope of Integration, more it becomes to isolate failures to specific component or system, which may leads to increased risk.
- Integration testing should normally be integral rather than big bang, in order to reduce the risk of late defect discovery
- Non functional characteristics (e.g. performance) may be included in Integration Testing

Function Testing

- It is used to detect discrepancies between a program's functional specification and the actual behavior of an application.
- The goal of function testing is to verify whether your product meets the intended functional specifications laid out the development documentation.
- When a discrepancy is detected, either the program or the specification is incorrect.
- All the black box methods are applicable to function based testing

System Testing

- It is concerned with the behavior of whole system as defined by the scope of development project
- It includes both functional and non-functional requirement of system
- System testing falls within the scope of black box testing.
- On building the entire system, it needs to be tested against the system specification.
- An Independent testing team may carry out System Testing

System Testing Types

- Usability testing
- Performance Testing
- Load Testing
- Stress Testing
- Security Testing
- Configuration Testing
- Compatibility Testing
- Installability Testing
- Recovery Testing
- Availability Testing
- Volume Testing

Usability Testing

- The typical aim of usability testing is to cause the application to fail to meet its usability requirements so that the underlying defects can be identified, analyzed, fixed, and prevented in the future.

Performance Testing

- Performance testing is testing to ensure that the application response in the limit set by the user.

Stress Testing

- Subject the system to extreme pressure in a short span.
- E.g Simultaneous log-on of 500 users
  - Saturation load of transactions

Configuration Testing

- Configuration testing is the process of checking the operation of the software you are testing with all these various types of hardware.

Compatibility Testing

- The purpose of compatibility testing is to evaluate how well software performs in a particular hardware, software, operating system, browser or network environment.

Acceptance Testing

- Acceptance testing may assess the system readiness for deployment and use
- The goal is to establish confidence in the system, parts of system or non-functional characteristics of the system
- Following are types of Acceptance Testing:
  - User Acceptance Testing
  - Operational Testing
  - Contract and Regulation Acceptance Testing
  - Alpha and Beta Testing

Objectives of Different Types of Testing

- In development Testing, main objective is *to cause as many failures as possible.*
- In Acceptance Testing, main objective is *to confirm that system work as expected.*
- In Maintenance Testing, main objective is *to make sure that no new errors have been introduced.*
- In Operational testing, main objective may be *to access system characteristics such as reliability and availability*.

Other Testing Types

Other than validation activities like unit, integration, system and acceptance we have the following other types of testing

- Mutation testing
- Progressive testing
- Regression testing
- Retesting
- Localization testing
- Internationalization testing

Mutation testing

- Mutation testing is a process of adding known faults intentionally, in a computer program to monitor the rate of detection and removal, and estimating the umber of faults remaining in the program. It is also called Be-bugging or fault injection.

Progressive Testing and Regressive Testing

- Most test cases, unless they are truly throw-away, begin as progressive test cases and eventually become regression test cases for the life of the product.

Regression Testing

- Regression testing is not another testing activity
- It is a re-execution of some or all of the tests developed for a specific testing activity for each build of the application
- Verify that changes or fixes have not introduced new problems
- It may be performed for each activity (e.g. unit test, function test, system test etc)

Regression Test

Testing software that has been tested before.

- Why?
  - Peripheral testing for bug fixes
  - Retesting the old version functionality with the new version

Retesting

Why retest?

- Because any software product that is actively used and supported must be changed from time to time, and every new version of a product should be retested

Localization Testing

- The process of adapting software to a specific locale, taking into account, its language, dialect, local conventions and culture is called localization.

Internationalization Testing

- The process of designing an application so that it can be adapted to various languages and regions without engineering changes.

Test Types : The Target of Testing

- **Testing of functions (functional testing)**
  - It is the testing of “what” the system does
  - Functional testing considers external behavior of the system
  - Functional testing may be performed at all test levels
- **Testing of software product characteristics (non-functional testing)**
  - It is the testing of “How” the system works
  - Nonfunctional testing describes the test required to measure characteristics of systems and s/w that can be quantified on varying scale
  - Non-functional testing may be performed at all levels

Test Types : The Target of Testing

Testing of software structure/architecture (structural testing)

- Structural testing is used in order to help measure the thoroughness of testing through assessment of coverage of a type of structure
- Structural testing may be performed at all levels.

Testing related to changes (confirmation and regression testing)

- When a defect is detected and fixed then the software should be retested to confirm that the original defects has been successfully removed. This is called Confirmation testing
- Regression Testing is the repeated testing of an already tested program, after modification, to discover any defects as a result of changes.
- Regression Testing may be performed at all levels.

Part-6 : Software Development & STLC

Software Development and Software Testing Life Cycle

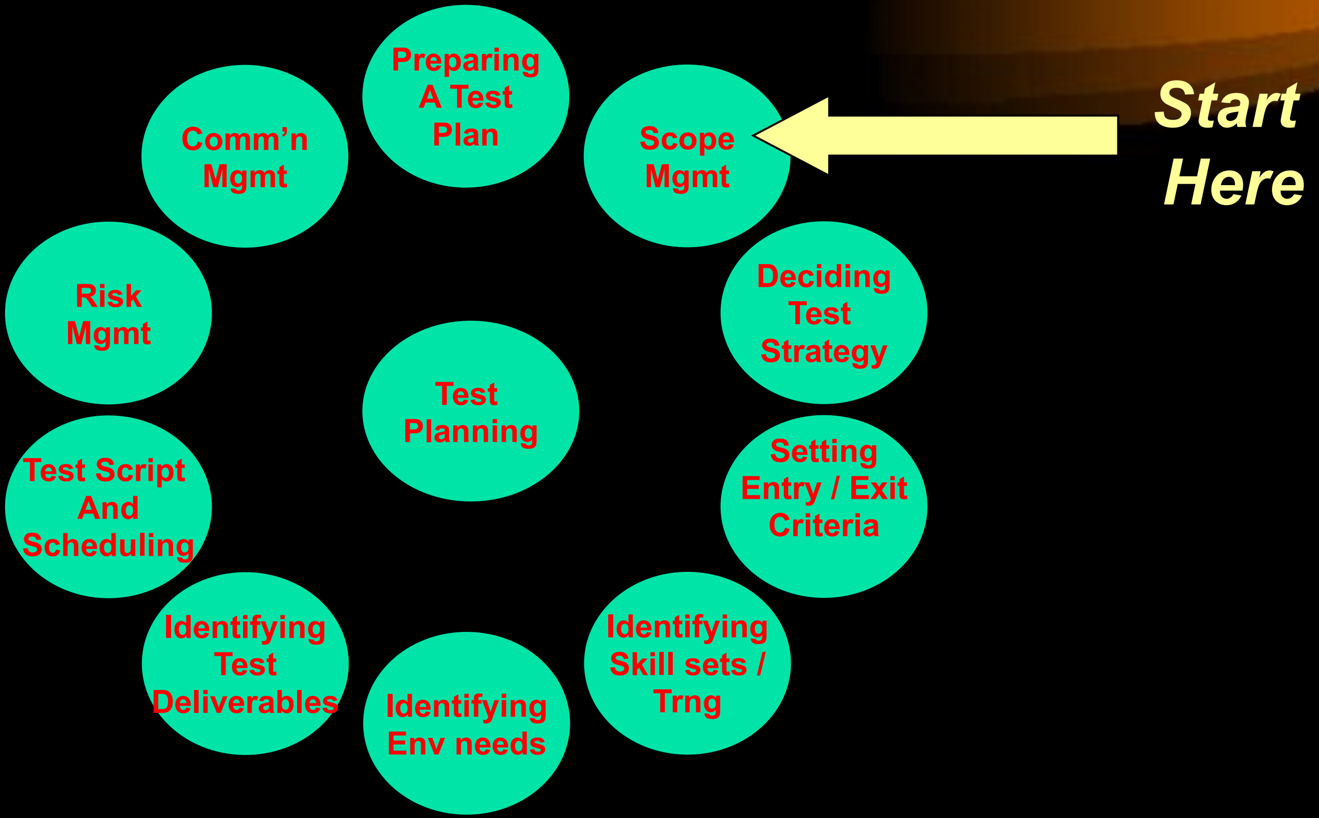
Test Planning

- It is the process of defining a testing project such that it can be properly measured and controlled

- It includes test plan, test strategy, test requirements and testing resources

Software Testing Genius

Parts of Test Planning



Test Design

- It is the process of defining test procedures and test cases that verify that the test requirements are met

- Specify the test procedures and test cases that are easy to implement, easy to maintain and effectively verify the test requirements.

Test Design Techniques

- During Test Design, the test basis documentation is analyzed in order to determine what to test i.e test condition
- A test condition is defined as an item that could be verified by one or more test cases
- Test case consist of a set of input values, execution preconditions, expected results and expected post conditions; are developed and described by using test design techniques
- Expected results should ideally be defined prior to test execution
- Test cases are put in an executable order, this is test procedure specification
- If tests are run using test execution tool, the sequence of actions is specified in a test script
- The test procedures and automated test scripts are subsequently formed into a test execution schedule

Categories of Test Design Techniques

- **Specification (White-box) Based techniques**
  - Models, either formal or informal; are used for the specification of problem to be solved
  - From these models test cases can be derived systematically
- **Structure (Black-box) Based techniques**
  - Information about how the software is constructed is used to derive the test cases
  - The extent of coverage of the software can be measured for existing test cases and further test cases can be derived systematically
- **Experienced Based techniques**
  - The knowledge and experience of people are used to derive test cases
  - Knowledge about likely defects and their distribution

Categories of Test Design Techniques

Specification based/White Box techniques

- Equivalence Partitioning
- Boundary value analysis
- Decision Table (Cause and Effect) Testing
- Use case Testing

Structure/Black Box based techniques

- Code Coverage
- Branch Coverage
- Statement Coverage

100% decision coverage guarantees 100% statement coverage, but not vice-versa

Test Management

- Test Planning is a continuous activity and is performed in all the life cycle processes and activities

- **Test Planning activities includes:**
  - Defining the overall approach
  - Integrating and coordinating the testing activities into software life cycle activities
  - Assigning resources for different tasks defined
  - Defining the amount, level of detail, structure and templates for test documentation
  - Selecting metrics for monitoring and controlling test preparation
  - Making decisions about what to test, what roles will perform the test activities, when and how test activities should be done, how the test results will be evaluated and when to stop the testing

Test Management

- **Exit Criteria** – Defines when to stop testing

- **Exit criteria may consist of**
  - Thoroughness measures, such as coverage of code, functionality or risk
  - Estimates of defect density or reliability measures
  - Cost
  - Residual risk
  - Schedules such as those based on time to market

Risk and Testing

- Risk is defined as “combination of likelihood and its impact it would have on the user.
- The level of risk will be determined by the likelihood of an adverse event happening and the impact
- Risk associated with testing can be classified in two categories:
- **Project Risk:** The risks that surround the project’s capability to deliver its objective
- **Product Risk:** Potential failures in the software or system are known as product risk, as they are a risk to the quality of the product

Project Risk Objectives

- **Suppliers Issues**
  - Failure of a third party
  - Contractual Issues
- **Organizational Factors**
  - Skill and staff shortage
  - Personal and training issues
  - Potential issues, such as problem with testers communication, failure to follow up the information found in Testing
  - Improper attitude towards testing
- **Technical Issues**
  - Problem in defining the right requirement
  - The extent that requirements can be met given existing constraints
  - Quality of design, code and tests

Product Risk Objectives

- **Product Risks Objective**
  - Error prone software delivered
  - Potential that the software/hardware could cause harm to company/individual
  - Poor software characteristics
  - Software that does not perform its intended functions

- A risk based approach to testing provides proactive opportunities to reduce the levels of product risks, starting in the initial stages of project

Test Development

- It is the process of creating test procedures and test cases that verify the test requirements
- Automated Testing using Tools
- Manual Testing

Test Basis

- The basis of test is the source material (of the product under test) that provide the stimulus for the test. In other words, it is the area targeted as the potential source of an error:
- Requirements-based tests are based on the requirements document
- Function-based tests are based on the functional design specification
- Internal-based tests are based on the internal design specification or code.
- Function-based and internal-based tests will fail to detect situations where requirements are not met. Internal-based tests will fail to detect errors in functionality

Test

- An activity in which a system or component is executed under specified conditions, the result are observed or recorded, and an evaluation is made of some aspect of the system or component.
- A set of one or more test cases.

Two basic requirements for all validation tests

- Definition of result
  - A necessary part of a test case is a definition of the expected output or result.
- Repeatability

Use Case

- To arrive at use cases, review the requirement statements; extract noun and verb pairs as use case "candidates".
  - **Following are few use case candidates.**

  - Organize photos into galleries
  - Review thumbnails
  - Expand thumbnail
  - Contact photographer
  - Upload photos
  - Include bio and contact info

- One use case is expanded into "actors" and "steps" to give you a feel for the level of detail in a use case.

Use Case

Actor	Step
Photographer	Selects photo to be uploaded.
Photographer	Selects gallery that photo should be uploaded to or creates a new gallery.
Photographer	Provides photo details such as camera, f-stop, shutter speed, focal length and artistic comments.
Photographer	Reviews posting.
Photographer	Changes or approves the posting.
Photographer	Reviews posting on website.
Photographer	Changes or deletes posting, if necessary.

Use Case

- Following matrix can be used to "trace" requirement statements to this use case.

Requirement ID	**Use Case**
SRS0006	Upload Photos

- The above trace-ability matrix tells that the use case "Upload Photos" satisfies the requirement SR0006 (Easy to upload photos, create galleries and enter info about the photo).

Use Case

Actor	Step	Rqt ID
Photographer	Selects photo to be uploaded.	SRS0006
Photographer	Selects gallery that photo should be uploaded to or creates new gallery.	SRS0006
Photographer	Provides photo details such as camera, f-stop, shutter speed, focal length and artistic comments.	SRS0006
Photographer	Reviews posting.	SRS0006
Photographer	Changes or approves the posting.	SRS0006
Photographer	Reviews posting on website.	SRS0006
Photographer	Changes or deletes posting, if necessary.	SRS0006

Test Case

- A set of test inputs, execution conditions, and expected results developed for a particular objective
- The smallest entity that is always executed as unit, from beginning to end
- A test case is a document that describes an input, action, or event and an expected response, to determine if a feature of an application is working correctly
- A test case should contain particulars such as test case identifier, test case name, objective, test conditions/setup, input data requirements, steps, and expected results
- Test case may also include prerequisites

Test Case

- A "test case" is another name for "scenario". It is a particular situation to be tested, the *objective* or *goal* of the test script. For example, in an ATM banking application, a typical scenario would be: "Customer deposits cheque for $1000 to an account that has a balance of $150, and then attempts to withdraw $200".

- Every Test Case has a *goal*; that is, the function to be tested. Quite often the goal will begin with the words "To verify that ..." and the rest of the goal is a straight copy of the functional test defined in the **Traceability Matrix**. In the banking example above, the goal might be worded as "To verify that error message 63 ('Insufficient cleared funds available') is displayed when the customer deposits a cheque for $1000 to an account with a balance of $150, and then attempts to withdraw $200".

Test Case Components

- The structure of test cases is one of the things that stays remarkably the same regardless of the technology being tested. The conditions to be tested may differ greatly from one technology to the next, but you still need to know three basic things about what you plan to test:
- **ID #:** This is a unique identifier for the test case. The identifier does not imply a sequential order of test execution in most cases. The test case ID can also be intelligent. For example, the test case ID of ORD001 could indicate a test case for the ordering process on the first web page.
- **Condition:** This is an event that should produce an observable result. For example, in an e-commerce application, if the user selects an overnight shipping option, the correct charge should be added to the total of the transaction. A test designer would want to test all shipping options, with each option giving a different amount added to the transaction total.

Test Case Components

- **Procedure:** This is the process a tester needs to perform to invoke the condition and observe the results. A test case procedure should be limited to the steps needed to perform a single test case.
- **Expected Result:** This is the observable result from invoking a test condition. If you can’t observe a result, you can’t determine if a test passes or fails. In the previous example of an e-commerce shipping option, the expected results would be specifically defined according to the type of shipping the user selects.
- **Pass/Fail:** This is where the tester indicates the outcome of the test case. For the purpose of space, I typically use the same column to indicate both "pass" (P) and "fail" (F). In some situations, such as the regulated environment, simply indicating pass or fail is not enough information about the outcome of a test case to provide adequate documentation. For this reason, some people choose to also add a column for "Observed Results."

Sample Business rule

- **Defect Number Cross-reference:** If you identify a defect in the execution of a test case, this component of the test case gives you a way to link the test case to a specific defect report.

- A customer may select one of the following options for shipping when ordering products. The shipping cost will be based on product price before sales tax and the method of shipment according to the table below.

- If no shipping method is selected, the customer receives an error message, "Please select a shipping option." The ordering process cannot continue until the shipping option has been selected and confirmed.

Characteristics of a Good Test

- **An excellent test case satisfies the following criteria:**
  - It has a reasonable probability of catching an error
  - It is not redundant
  - It's the best of its breed
  - It is neither too simple nor too complex

Test Execution

- It is the process of running a set of test procedures against target software build of the application under test and logging the results.
- Automation

Test Evaluation

- It is the process of reviewing the results of a test to determine if the test criteria are being met.

What should our overall validation strategies be?

- Requirements-based tests should employ the black-box strategy. User requirements can be tested without knowledge of the internal design specification or the code.
- Function-based tests should employ the black-box strategy. Using the functional-design specification to design function-based tests is both necessary and sufficient.
- Internal-based tests must necessarily employ the white-box strategy. Tests can be formulated by using the functional design specification.

Part 7: Defect Management

What is definition of defect?

A flaw in a system or system component that causes the system or component to fail to perform its required function. - SEI

A defect, if encountered during execution, may cause a failure of the system.

The purpose of finding defects

- The purpose of finding defects is to get them fixed
- The prime benefit of testing is that it results in improved quality. Bugs get fixed

Why do defects occurs

- There are various reasons for the occurance of the faults; it may be due to
  - Poor documentation
  - Ambiguous or unclear requirements
  - Lack of programming skills
  - Due to increase in work pressure and assigned deadlines

Status associated with a defect

- **New:** When a bug is found/revealed for the first time, the software tester communicates it to his/her team leader in order to confirm if that is a valid bug. After getting confirmation from the test lead the software tester logs the bug and the status of “New” is assigned to the bug.

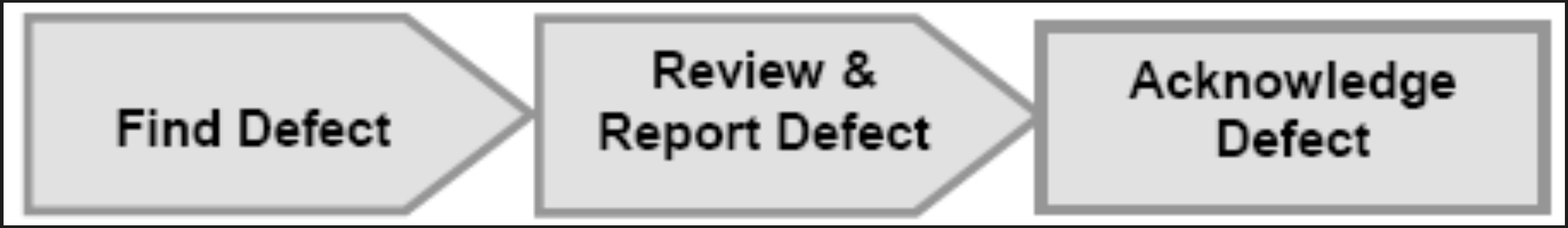
- **Assigned:** after the bug is reported as “New” it comes to the development team. The development team verifies if the bug is valid. If the bug is valid, development leader assigns it to a developer to fix it and a status of “assigned” is assigned to it.

- **Open:** Once the developer starts working on the bug, he/she changes the status of the bug to “Open” to indicate that he/she is working on it to find a solution.

Status associated with a defect

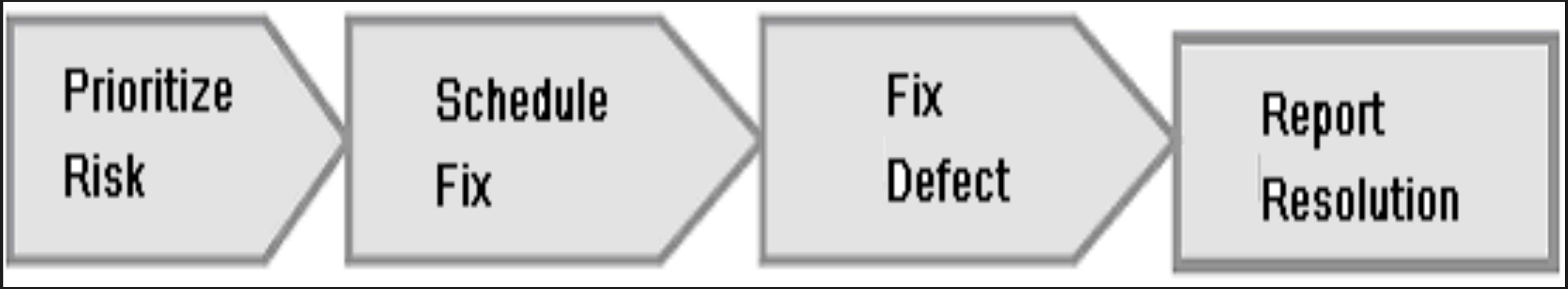
- **Fixed:** Once the developer makes necessary changes in the code, he/she marks the bug as “Fixed” and passes it over to the development Lead in order to pass it to the testing team.
- **Pending retest:** After the bug is fixed, it is passed back to the testing team to get retested and the status of “Pending Retest” is assigned to it.
- **Retest:** The testing team leader changes the status of the bug, which is previously marked with “Pending Retest” to “Retest” and assigns it to a tester for retesting.
- **Closed:** After the bug is assigned a status of Retest, it is again tested. If the problem is solved, the tester closes it and mark it with “closed” status.
- **Reopen**
- **Rejected**

Defect Discovery



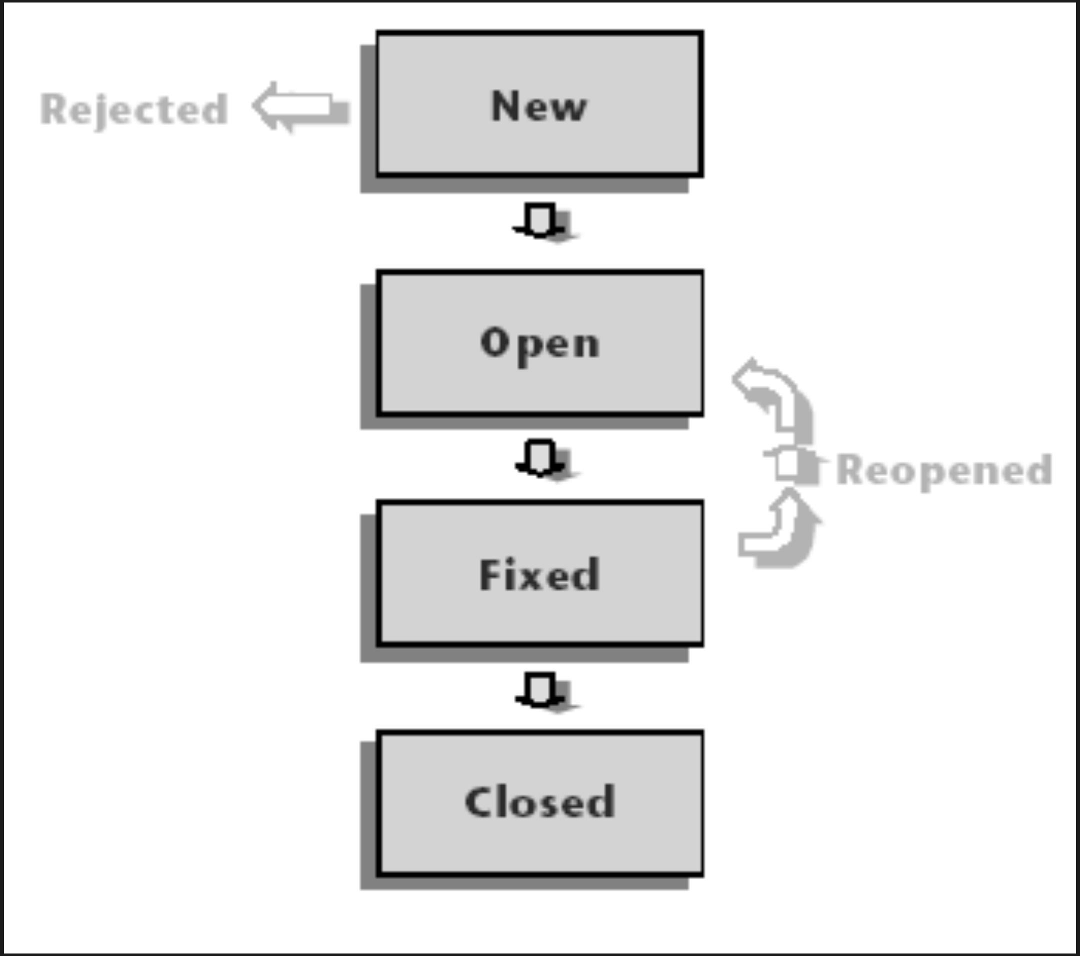
Defect Discovery Process

Defect Resolution



Defect Resolution Process

Defect Life Cycle



Defect Life Cycle

When a tester reports a Defect, it is tracked through the following stages: ***New*, *Open*, *Fixed*, and *Closed*.** A defect may also be ***Rejected***, or ***Reopened*** after it is fixed. A defect may be **Deferred** for a look at a later point of time.

By default a defect is assigned the status ***New*.**

A quality assurance or project manager reviews the defect, and determines whether or not to consider the defect for repair. If the defect is refused, it is assigned the status ***Rejected*.**

If the defect is accepted, the quality assurance or project manager determines a repair priority, changes its status to ***Open*,** and assigns it to a member of the development team.

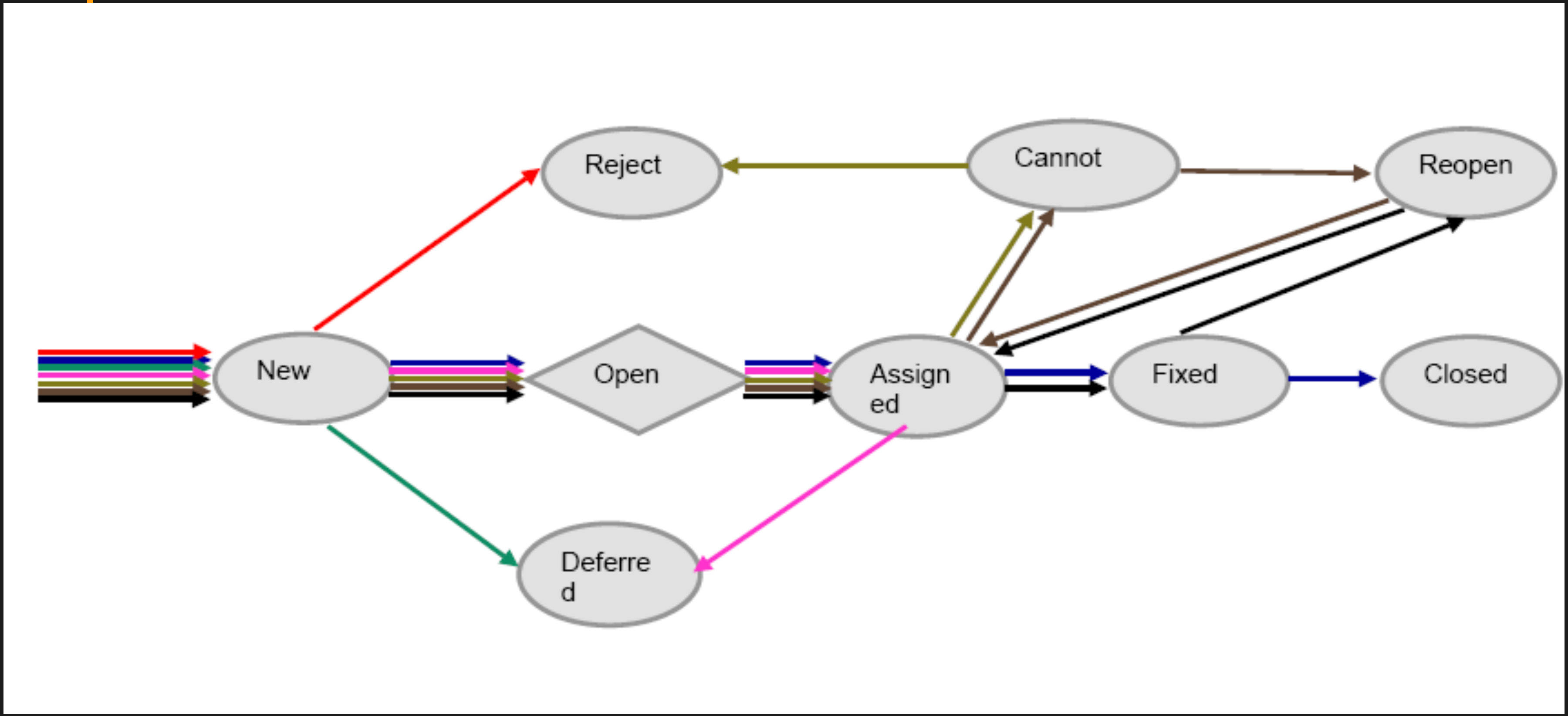
Defect Life Cycle

A developer repairs the defect and assigns it the status ***Fixed*.**

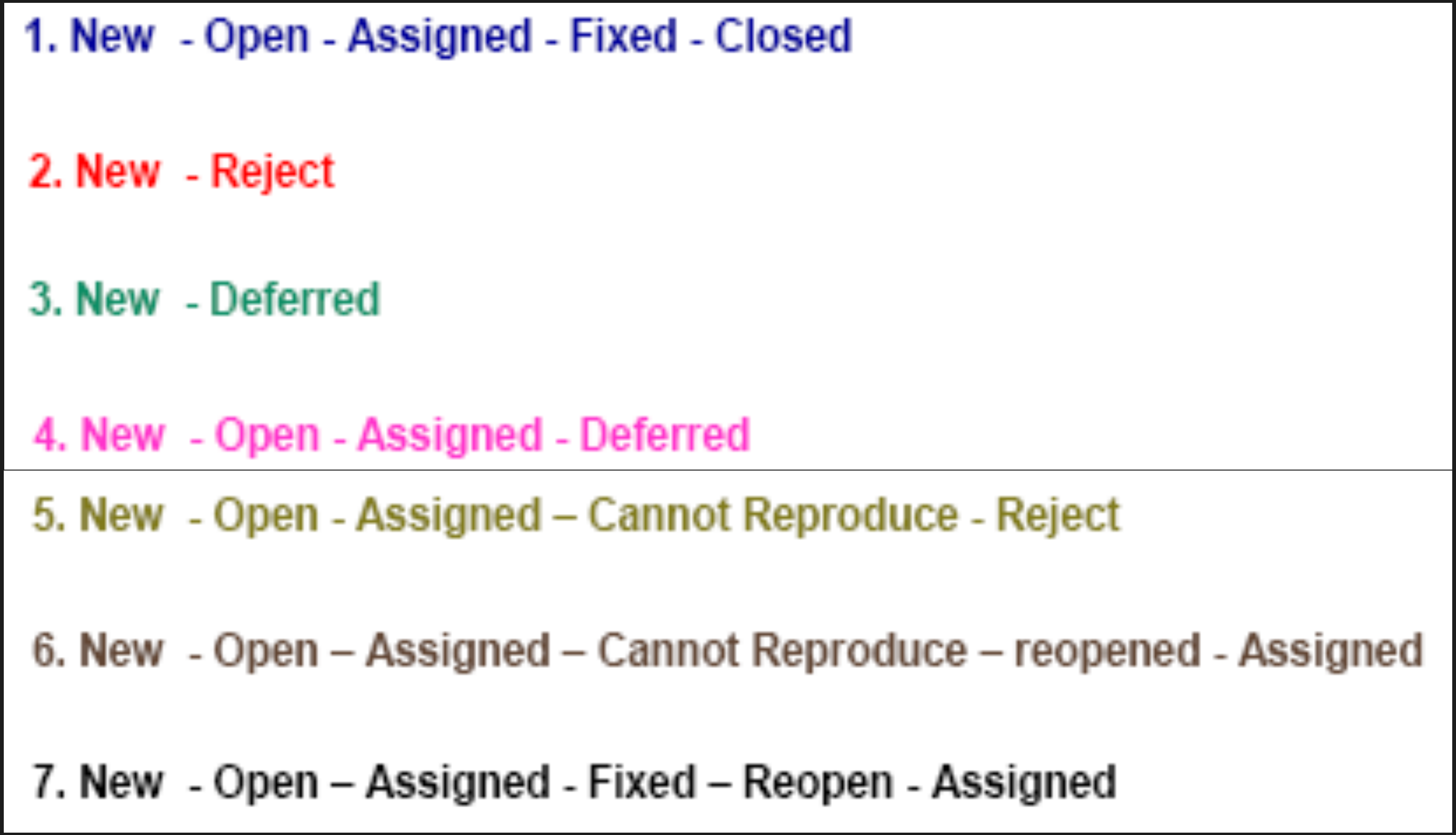
Tester retests the application, making sure that the defect does not recur. If the defect recurs, the quality assurance or project manager assigns it the status ***Reopened*.**

If the defect is actually repaired, it is assigned the status ***Closed*.**

Defect Life Cycle Paths

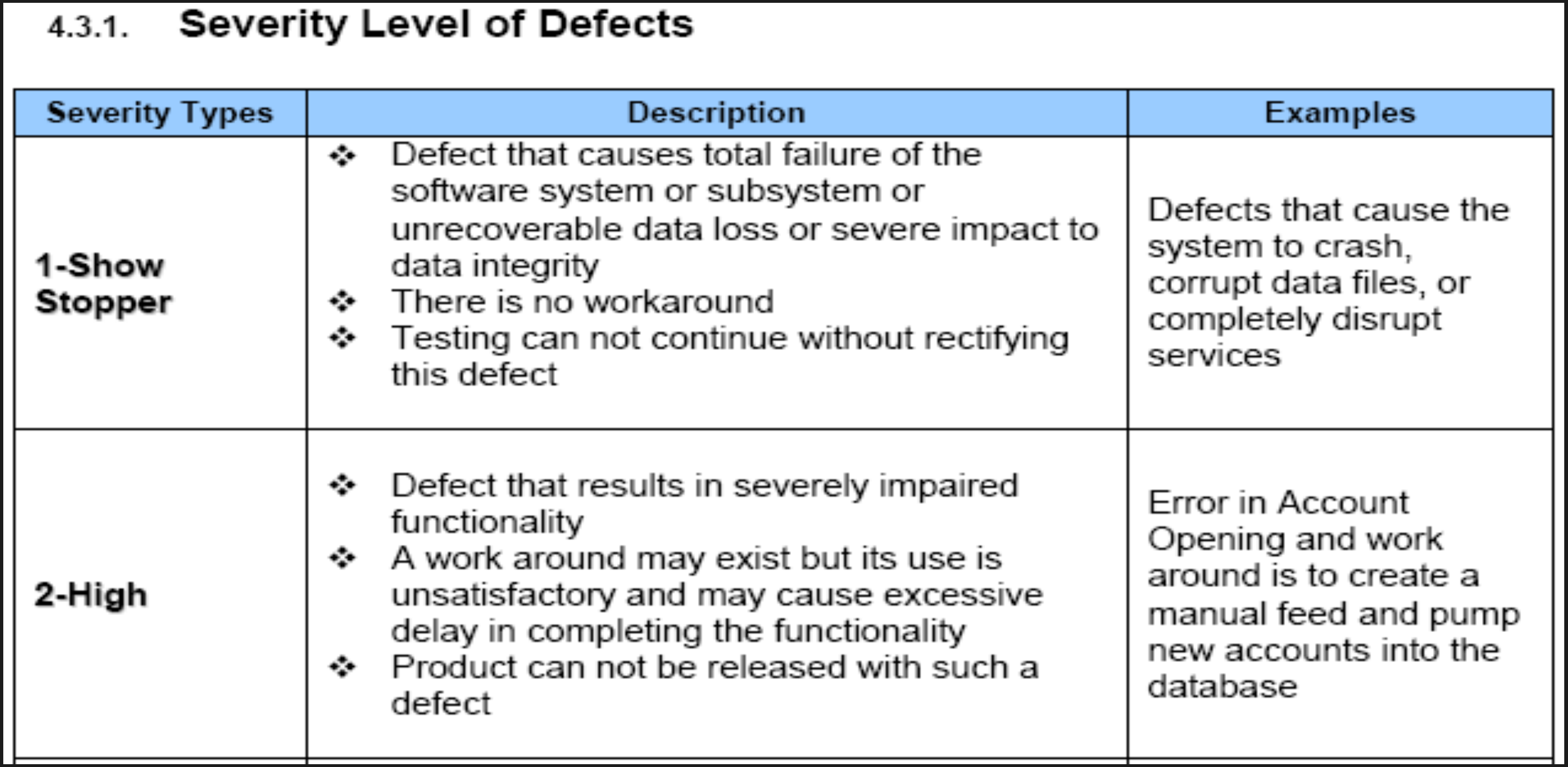


Defect Life Cycle Paths



1. New - Open - Assigned - Fixed - Closed
2. New - Reject
3. New - Deferred
4. New - Open - Assigned - Deferred
5. New - Open - Assigned – Cannot Reproduce - Reject
6. New - Open – Assigned – Cannot Reproduce – reopened - Assigned
7. New - Open – Assigned - Fixed – Reopen - Assigned

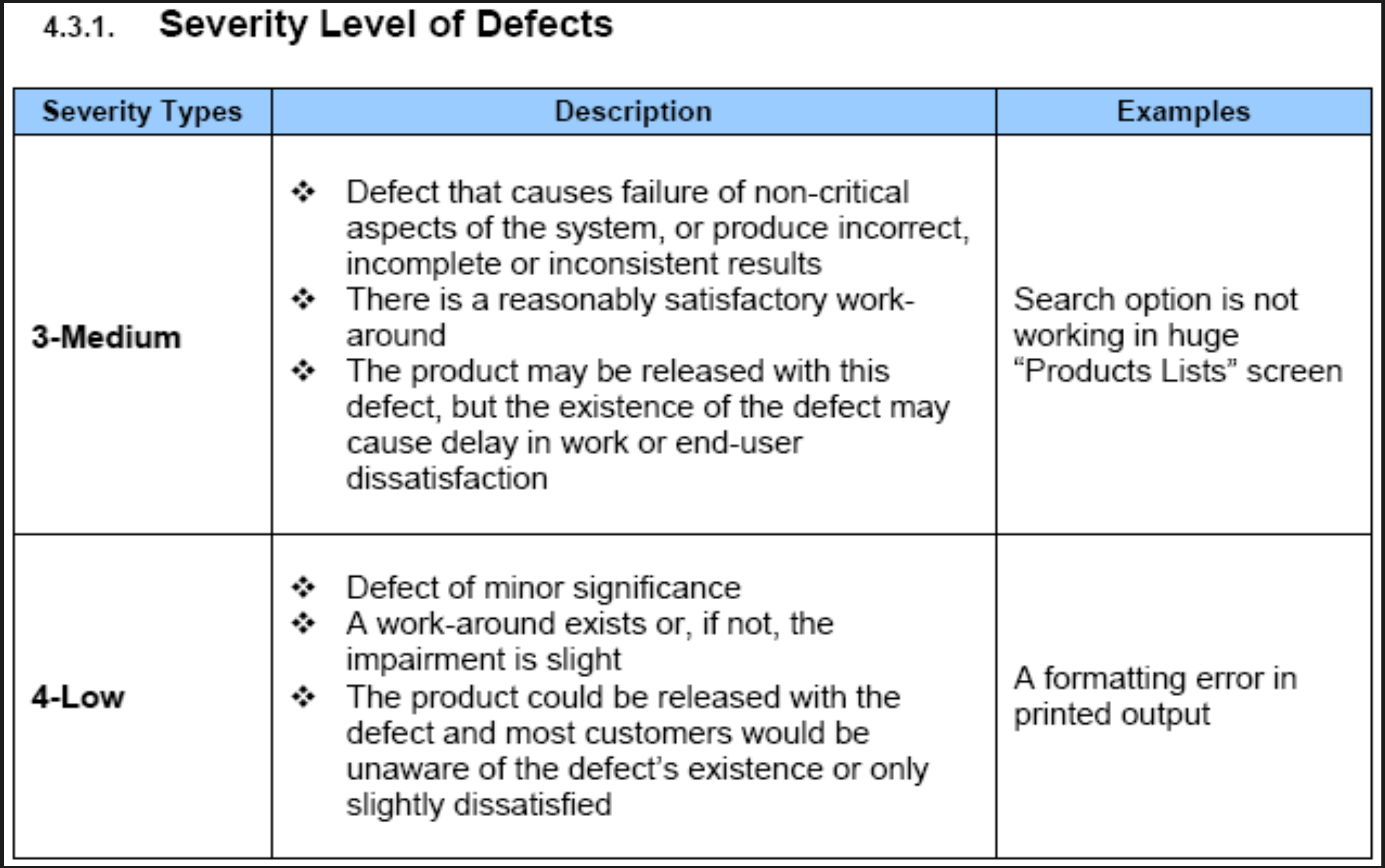
Defect Classification



4.3.1. **Severity Level of Defects**

Severity Types	Description	Examples
1-Show Stopper	❖ Defect that causes total failure of the software system or subsystem or unrecoverable data loss or severe impact to data integrity ❖ There is no workaround ❖ Testing can not continue without rectifying this defect	Defects that cause the system to crash, corrupt data files, or completely disrupt services
2-High	❖ Defect that results in severely impaired functionality ❖ A work around may exist but its use is unsatisfactory and may cause excessive delay in completing the functionality ❖ Product can not be released with such a defect	Error in Account Opening and work around is to create a manual feed and pump new accounts into the database

Defect Classification

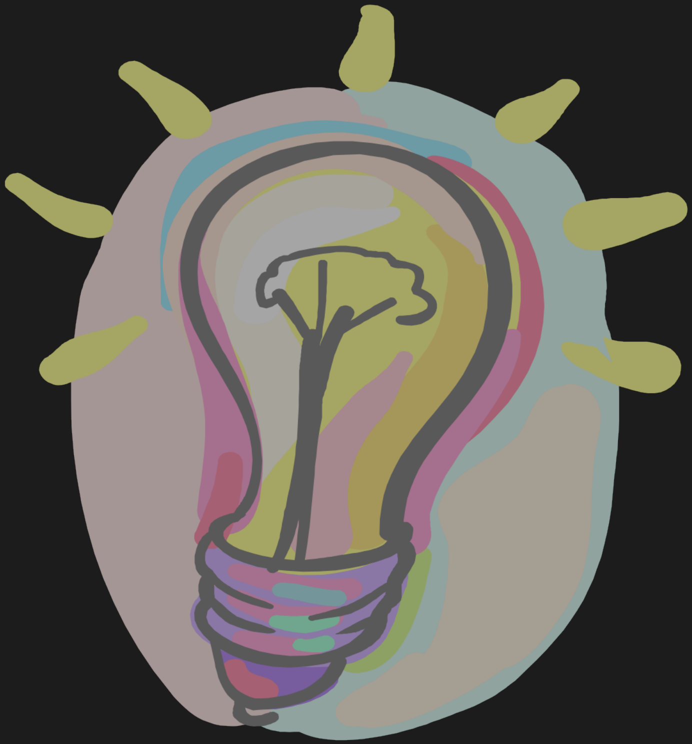


4.3.1. **Severity Level of Defects**

Severity Types	Description	Examples
3-Medium	❖ Defect that causes failure of non-critical aspects of the system, or produce incorrect, incomplete or inconsistent results ❖ There is a reasonably satisfactory work-around ❖ The product may be released with this defect, but the existence of the defect may cause delay in work or end-user dissatisfaction	Search option is not working in huge "Products Lists" screen
4-Low	❖ Defect of minor significance ❖ A work-around exists or, if not, the impairment is slight ❖ The product could be released with the defect and most customers would be unaware of the defect's existence or only slightly dissatisfied	A formatting error in printed output

How many testers do we need to change a light bulb?

- **None.** Testers just noticed that the room was dark.
- Testers don't fix the problems, they just find them



What Do You Do When You Find a defect?

- Report a defect
- The point of writing Problem Reports is to get bugs fixed.

Some typical defect report fields

- Summary
- Date reported
- Detailed description
- Assigned to
- Severity
- Detected in Version
- Priority
- System Info
- Status
- Reproducible
- Detected by
- Screen prints, logs, etc.

Who reads the defect reports?

- Project Manager
- Executives
- Development
- Customer Support
- Marketing
- Quality Assurance
- Any member of the Project Team

Part 8: Test Automation

Principles of Test Automation 1: Choose carefully what to automate

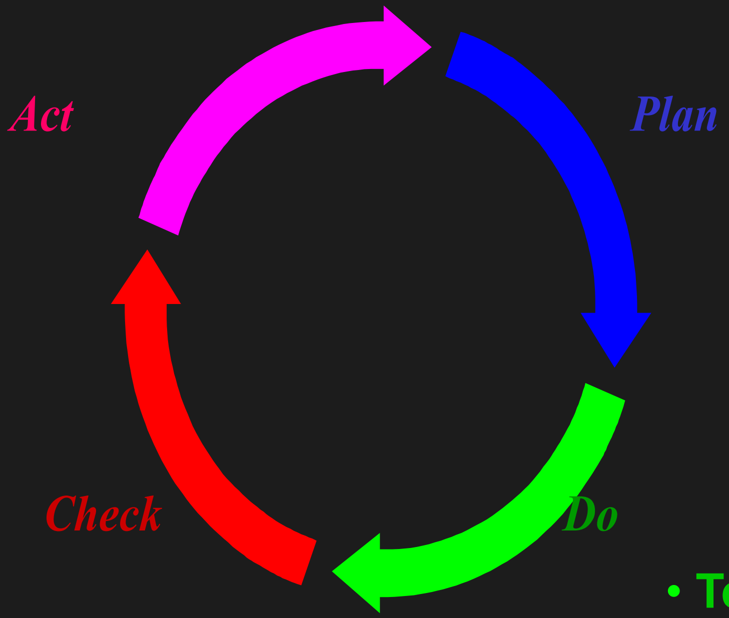
- Automate tests that absolutely cannot be done manually (e.g., stress test)
- Automate those tests that are part of the regression bed / smoke test
- Automate tests that are run repeatedly (e.g., tests that are run at least 20 times over the next quarter)
- Automate tests with no manual intervention
- Automate tests for standards (e.g., JDBC compliance)
- Automate tests that are covering areas "static" but important areas
- Automate tests that are covering areas "static" but important areas

Part 8: Test Automation

Principles of Test Automation
# 1: Choose carefully what to automate

- Automate tests for highly visible areas
- Minimize automating change-prone areas
- Between GUI and non-GUI portion automation, go for automating non-GUI portions first
- Automate tests for dependencies to catch ripple effects early
- Automate areas where multiple combos are possible (pros and cons)
- Automate areas that are re-usable
- Automate “easy areas” to show low hanging fruits

Principles of Test Automation
# 2: Ensure Automation Covers Full Circle



- Test Planning
- Automation Planning

- Test Capture
- Test Execution
- Results Comparison

- Automatic Analysis
- Fish Bone Diagrams
- Problem Identification

- Corrective Action Implementation
- Automatic Rollover to next runs
- Incorporation into Regression

Principles of Test Automation
# 3: Choose Proper Automation Tool

- Compatibility to Platform
- Portability across platforms
- Integration with TCDB, DR and SCM
- 2-way mapping to source code (may not be possible in services)
- Scripting Language
- Compatible to Multiple Programming Environments
- Configurability
- Test Case Reusability
- Selective Execution
- Smart Comparison
- Reliable Support
- Current documentation

Principles of Test Automation
# 4: Plan for Infrastructure

- Resources for Installation
- Resources for ongoing execution
- People Resources

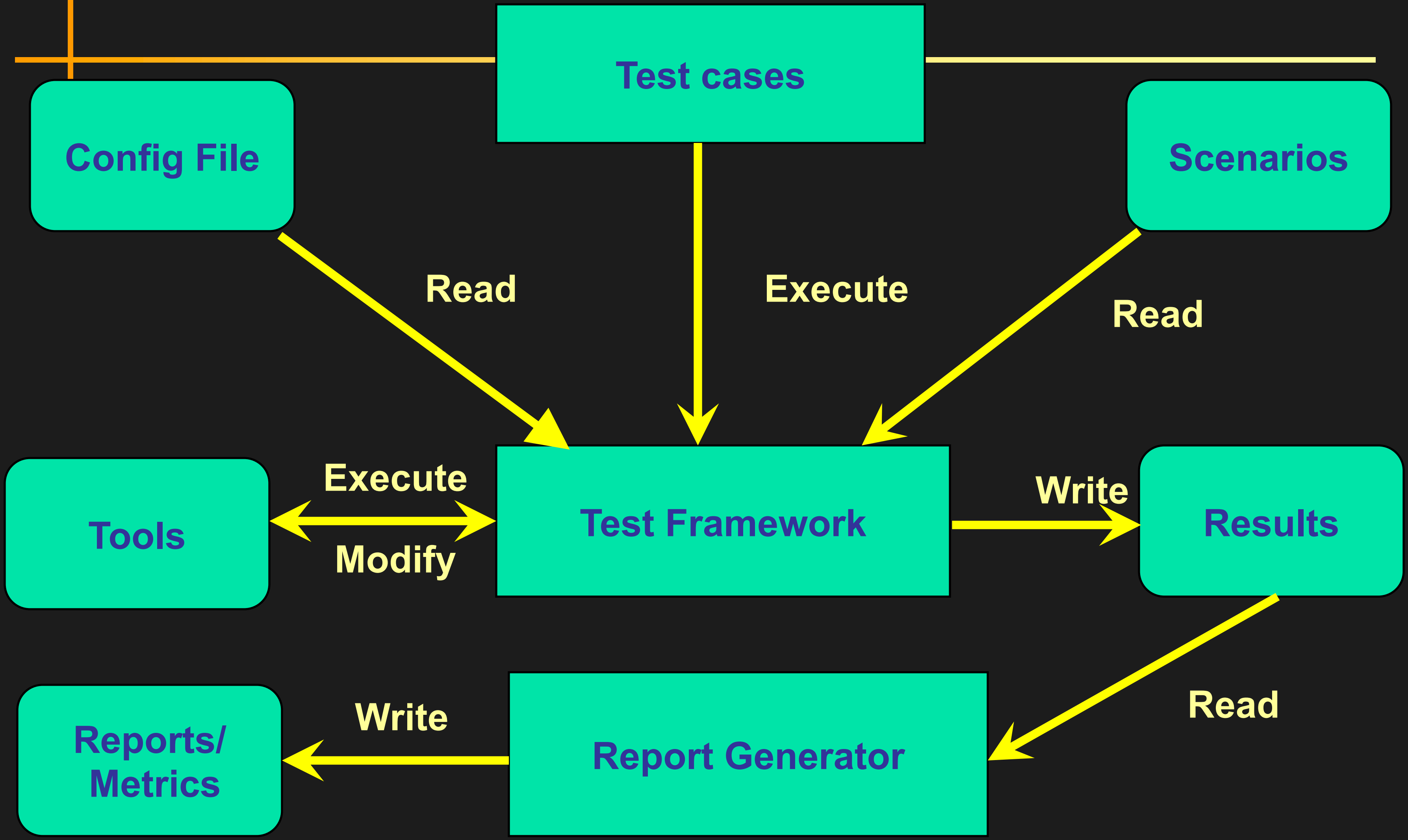
Principles of Test Automation
# 5: Account for Gestation Period

- Training
- Development
- Testing the Tests
- Sync-ing with product version changes

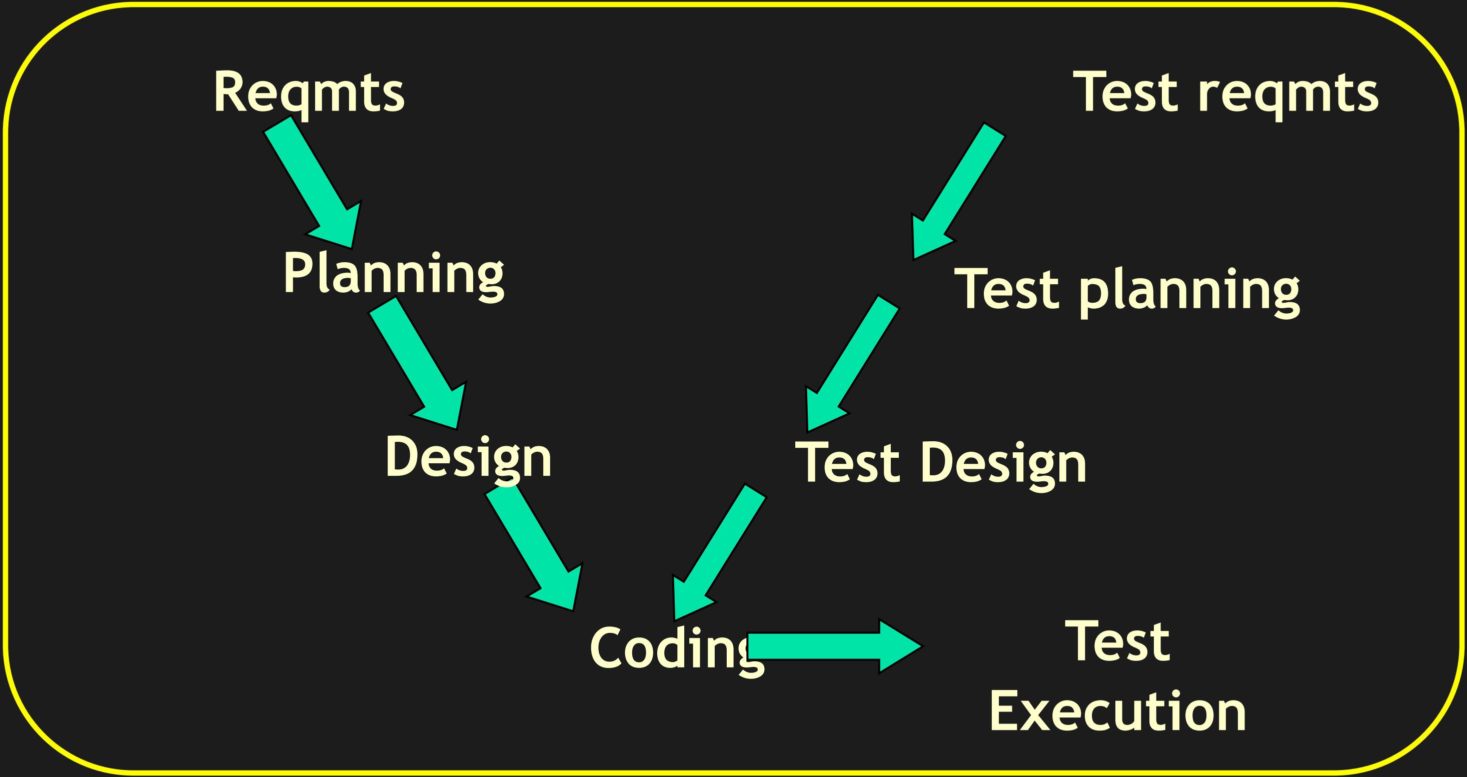
Principles of Test Automation
# 6: Run a Trial & Calibrate the Tool

- Start small
- Don’t try to automate everything at the same time
- Allow time for evolving standards

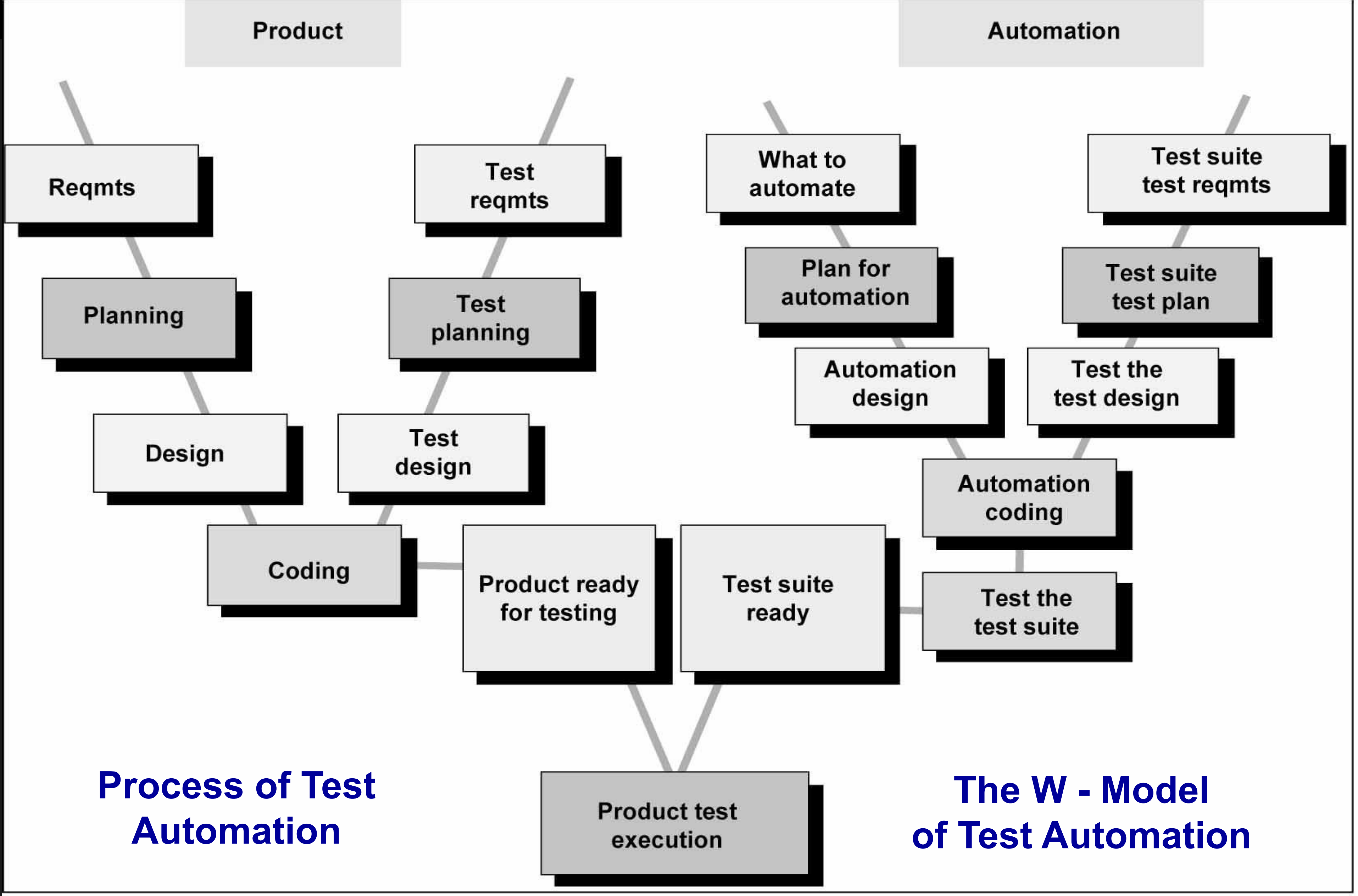
Test Automation Components



Process of Test Automation



The V - Model of Software Development



Process of Test Automation

The W - Model of Test Automation

Process of Test Automation

- The work on automation should go in parallel with product development
- The requirements for automation spans over to multiple phases for multiple releases like the product requirements
- The automation can be carried out in parallel with test design
- Sometimes the test cases need to be modified to suit the automation requirements; where automation exist the test cases need not be elaborate
- Where a dedicated team exist for automation, the schedule for automation can be independent of product releases, with some (tested) deliverables marked for each product release

Common Experiences in Test Automation

- There are plenty of tools available and rarely does one tool meet **all** the requirements
- The test tools are expensive (both in upfront costs and running costs)
- Test tools also require good amount of training and only few vendors available for training
- Training may not always keep pace with new versions of the tools
- Test tools expect the users to learn new language/scripts and may not use standard languages/scripts
- Deploying a test tool requires equal amount of effort as deploying a new product in a company – never underestimate the effort and pain involved!

Common Experiences in Test Automation

- Migrating from one test tool to another may be difficult and requires good amount of effort
- Test tools are one generation behind and may not provide backward / forward compatibility (eg. JAVA SDK support)
- Good number of test tools requires their libraries linked with product binaries – Causes portions of the testing to be repeated after those libraries are removed (eg. Performance)
- Test tools are not 100% cross platform – They are supported only on some platforms and the sources generated from these tools may not be compatible on other
- Developing sharewares/public domain test tools may not get same amount of participation/involvement/support as of standards/products (eg. As against Linux)

Common Experiences in Test Automation

The experiences

• Test tools may not go through same amount of evaluation for new requirements (eg Year 2000, 508)

•The test tools increases the system requirements and requires the H/W and S/W to be upgraded at compile/run-time

• The test tools are capable of testing only the product, not the impact because of the product/test tool to the system or network

• Good number of test tools can’t differentiate between a product failure and the test suite failure – Causing increased analysis time and manual testing

Common Experiences in Test Automation

The experiences

•The test tools may not provide good degree of trouble shooting / debug/error messages to help in analysis – Resulting in increased "printf"/log messages in the test suite

• The test tools determine the results based on messages and screen co-ordinates at run-time – Intelligence needed to proactively find out the changes

Common Pitfalls in Test Automation

- Automation shouldn’t be considered as stop-gap arrangement to engage test engineers (when no test execution, do automation!). Test Automation, like any other project, should start with the end in mind
- A separate team in the organization looking at automation requirements, tool evaluation and developing generic test suites would add more value (may not always apply to testing services organization)
- Automation doesn’t stop with automating the test cases alone. The test suite needs to be linked with other tools for increased effectiveness (e.g., Test case database, Defect filing, auto mails, preparing automatic reports, etc)
- Automation doesn’t stop with recording & playing back the user commands; Automated tool should be intelligent enough to say what was expected, why a test case failed and give manual steps to reproduce the problem

Thank You