Software Testing

- Introduction to Software Testing, Principles of Testing, Testing Life Cycle, Phases of Testing, Types of Testing, Verification & Validation, Defect Management, Defect Life Cycle, Bug Reporting, GUI Testing, Test Management and Automation.

Introduction

- It is a process used to identify the **correctness, completeness and the quality of developed software.**
- It is also helps to identify errors, gaps or missing requirements.
- It can be done either manually or using some software tools.

Strategic Approach

- General characteristics of strategic testing:
  - To perform effective testing, the software team should conduct technical reviews.
  - Testing begins at component level and work toward the integration of entire computer based system.
  - Different testing techniques are appropriate at different points in time.
  - Testing is done by developer of software and for large projects by an independent test groups

- Testing and debugging are different activities, but debugging must accommodate in any test strategy.

- In other words, **software testing is verification and validation process**.

Principles of Software Testing

- There are seven principles of software testing

1. Testing shows presence of defects
2. Exhaustive testing is impossible
3. Early testing
4. Defect Clustering
5. Pesticide Paradox
6. Testing is context dependent
7. Absence of Error-fallacy

Testing shows presence of defects

- The goal of testing is to make software fail. Sufficient testing reduces the presence of defects.
- The effective testing can show **the defects that are present in software, but it can not prove that the software is defect free.**
- Even after testing the application or software thoroughly we cant say that the software is defect free.
- An effective testing always reduce the number of defects remaining in the software
  - But even if **no defect is found in testing, it doesn't prove that software is defect free**.
- **Testing talks about presence of defects and it does not talk about absence of defects.**

Exhaustive testing is impossible

- Testing all functionalities using all valid and invalid inputs and preconditions is known as exhaustive testing.

- For Ex: In an application, on one screen there are 15 fields having 5 possible values, then too many valid combinations….

- Why it is impossible to achieve it?
  - Assume we have to test input field: which may take billion range values
  - If we try all conditions, software execution time and cost will raise
  - **Instead, we can take risk and priorities into consideration**.

- Accessing and managing the risk areas be most important for testing any project.

Early testing

- **Defects detected in the early phases of SDLC are less expensive to fix**. So conducting early testing reduces the cost of fixing defects.
    - If you have identified incorrect requirement in the requirement gathering phase
    - If you have identified a bug in the fully developed functionality
- **It is cheaper to change incorrect requirement compared to fixing bug in fully developed functionality**

Defect Clustering

- **Small module of functionality contains most of the bug**s or it has the most operational failures.

- As per the Pareto principle(80-20 rule),
  - 80% of issue comes from 20% of modules,
  - So we do emphasis testing on the 20% of the module, where we can face 80% of bugs

Pesticide Paradox

- It is the process of **repeating the same test cases again and again**,
- It will not find the new bugs.
- So, to overcome this pesticide paradox, **it is necessary to review the test cases regularly and add or update them to find more defects**

Testing is context dependent

- **Testing approach depends on the context we develop.**
- We do test the software differently in different contexts.
- For Ex: Online banking application requires a different approach of testing compared to e-commerce site.

Absence of Error-fallacy

- **99% of bug free software may still be unusable**, if we incorporated wrong requirement into the software and the software is not addressing the business needs.

- The software which we built should not be only bug free software, it must be addressing the business needs.

Software Testing Strategy

- Spiral Model.
  - **The software is developed by moving inward in the spiral.**
  - **Testing is conducted by moving outward in the spiral.**

- System Engineering: defines the roles of the software and leads to requirement analysis (performance constraints, validations criteria and functions of software)

- Unit testing: focuses on testing each module/component independently based on implementation.
  - Units internal code/structure
  - Path testing (test cases of code, control structure etc)
  - Ensures complete coverage of the unit independently.

- Integration testing: **Components are integrated together step by step to form a complete software**.
  - Software architecture is tested.
  - Focus is on construction of software

- Validation testing: **Requirements established are validated against the development software.**
  - The behaviour of the software
  - Performance requirements
  - Validation criteria

- System Testing : The software is tested with other system elements as a whole

Unit Testing

- It is a process of **taking a module and testing it in isolation** from the rest of the software and
- **comparing the actual results with the results defined in the specifications and design of mo**dule.

- Why to test each unit independently?
  - Fault Isolation and debugging becomes easier when components are tested independently .

- What to test during unit testing?

  - Module interfaces
    - Tested to check the **correct flow of information in and out of the module**.

    (Parameter passing is correct or not, Result generated)

  - Local data structure
    - Are examined to ensure that the intermediate results or **data is maintained or stored correctly**.

  - Independent/basic paths
    - Tested to ensure all statements within the module executed at least once during testing
    - (decision statements/control paths)

  - Boundary conditions
    - To ensure output/computation at boundary values is correct

  - Internal logic
    - Loops, precedency of operators, comparison between different data types

  - Error handing paths

- Problems with unit testing:
  - A component is not a stand alone unit, so
    - How to test it without anything to call it?
    - How to test it without any module to be called by it?

Drivers and stubs

- Drivers: (calls the test module)
  - It is main module that accepts the test case data
  - Accepts test case data
  - Passes data to the component to be tested and prints relevant results

- Stubs: (called by the test module)
  - Subordinate modules that are called by module to be tested
  - It is a dummy sub-program that does minimal data manipulation
  - Provides verification of entry and returns the control to module under testing.

- Scaffholding:
  - the overhead code.
  - Refer to drivers and stubs.
  - Refer to test harness (automatically generated overhead code)
  - Test harness
    - Used to automatically generate scaffholding
    - It allows us to test our module in an isolated environment by simulating the rest of the software functionalities.

Integration Testing

- Determining the correctness of the interface is the focus of interaction testing.

- Why integration tests required?
  - Data may be lost during interfacing
  - Global data may cause problems
  - sub functions may not work properly when combined together

- Methods to integrate and Test:
  - Unit tested components are taken one by one and integrated incrementally (debugging and fault isolation becomes easier)

- Types of integration:
  - Top down integration:
  - Bottom up integration
  - Sandwich integration

- Methods to integrate and Test:
  - Unit tested components are taken one by one and integrated incrementally (debugging and fault isolation becomes easier)

- Types of integration:
  - Top down integration:
    - Calling modules are always available, no drivers required
    - Stubs are required
  - Bottom up integration
    - Calling module are not available, drivers are required
    - Stubs are not required
  - Sandwich integration

Top-down integration

- Move down in control hierarchy.

- Start with main module and integrate the sub modules
  - Depth first integration: integrate the units/components on a major control path.
  - Breadth first integration: combine all the components that are directly subordinate to a given module

Top-down integration

- Integration steps:
  - Starting with the main control module, stubs are substituted for all components subordinate to it.
  - Depending on type of integration, stubs are replaced one at a time with actual components.
  - Test are conducted as each component integrated.
  - On completion, another stub replaced with actual component
  - Regression testing ensures no new error.
- Problems:
  - Stubs lead to overhead code.

Bottom-up integration

- Begin testing the module at lowest levels
- Drivers required but no stubs needed
  - The processing at the lower levels are always available

- Integration steps:
  - Low level components are combined into clusters that perform specific software function.
  - Drivers coordinate the test case input-output.
  - The cluster is tested
  - Driver is removed and the clusters are combined with new module moving up in the hierarchy

Regression Testing

- In addition of a new module as a part of integration testing, may cause problems with the function that previously worked flawlessly.

- What is Regression testing:
  - **Re-execution of some subset of test cases, that have already been conducted to ensure that the changes or addition of new modules does not create side-effects.**

- How to do?
  - automated or manually
  - Automatically: capture/playback tools
  - These tools enable software engineer to capture the test cases and their results for subsequent playback and comparison.

- What tests are conducted?
  - We need to examine the functionality of all the modules.
  - We have to focus on the functionality that is most likely to get affected by the newly integrated module.
  - Test only those software components that have been changed.

Validation Testing

- Focuses on user viable actions and user recognizable output from the system.
- **Validation succeeds when software function in a manner that is expected by the customer**
- **Validation criterion:**
  - Written at the time of SRS.
  - Acts as a contract between two parties.
  - A section in the SRS which describes the desired functionality that forms the basis of validation testing.

- How is validation achieved.
  - Through a series of tests that clearly demonstrates the existence of the functionality required by the user in software.

- What does validation testing ensure?
  - Functionality is achieved
  - Correct behaviour of the software
  - Performance constraints met.
  - Documents are correct
  - Non-functional requirements like usability, robustness, fault tolerance etc.

  A deficiency list may be created in case something is miisin/incorrect

System Testing

- It incorporates the software with other system elements.
- System testing ensure that all system elements have integrated well with the software and the entire system functions properly.
- It ensure full proof testing of entire system.

Types of System Testing

- Recovery testing
  - **Ensures that the system must recover form faults and resume processing with little or no down time.**

  (Fault tolerant system: Any processing fault should not bring down the entire processing system.)

  - Software must recover from faults and resume its normal processing within a specified time period.

  - How it is done?
    - Force the system to fail in different ways
    - And then we verify that the recovery of system and its functionality is done properly.

- Security testing:
  - Verifies that the protection mechanism built into the system will actually protest it for attacks.
  - Responsibility of tester
    - Acts as a intruder himself,
    - tester ensures that the cost of the attack is higher than the information achieved from the attack.

- Stress Testing:
  - It executes the system in manner that demands resources in abnormal quantity.

  - Example:
    - System can handle only 10 request per minute.
    - If we exceed the threshold, does the system fails?

    - **Excessive memory requirements, database requests, Simultaneous logins etc**

- Sensitivity testing:
  - Variation of stress testing
  - It attempts to uncover data combinations that are within the valid input domain, but which may cause improper functioning of the system.

- Performance Testing:
  - Aimed at testing the run-time performance of the software.
  - It is done throughout the software
  - Ensure that performance is monitored continuously beginning form the unit testing phase

- Deployment testing/configuration testing:
  - It tests the software in each environment in which it is to operate
  - It examines all kinds of installation procedures, documents required.

Acceptance Testing

- Conducted when software is developed **for a specific customer** and not for a large public/audience.

- Purpose:
  - Allows customer to validate all requirements.

- Process:
  - Customer tests the software
  - Customer gives the feedback
  - Negative feedback-modification are done
  - Final delivery of the software to the customer

Alpha and Beta Testing

- Conducted **when the product is developed for anonymous customer**.
- For large public
- Carried by the customer.

Alpha testing	Beta Testing
Done by the customer by the developers site	Done by the customer at the users/customers site
Conducted in controlled environment	Conducted in real time environment of the user Not under the developers control
Developer present?: Yes	Developer present? : No
Carried out before the release of the product to the customer	Carried out after the release of product to customer
Errors/failures are recorded	Failures are reported
White and black box testing	Black box testing

Verification and Validation

Verification	Validation
Are we building the system right?	Are we building the right system/software
It is a process of defining a system or component to determine if the products of a development phase satisfy the conditions imposed at start that phase	It is the process of evaluation a system/component during or at the end of development process to determine whether it satisfied the specified requirement
Applied during the development phase	Applied towards the end of development process

Verification	Validation
Includes manual testing: look and review the document	Program execution takes place: Requirements are validated
Activities involved: Reviews, meetings, inspections	Activities involved: testing techniques,
Carried out by internal quality assessment team	Carried out by testing team
Done prior to Validation	Done after Verification

- Both Verification and validation are complementary activities.

- If we are able to find more errors before execution (verification-minimizes errors in only phases of development), validation becomes easier

DEFECT MANAGMENT

- **What is Bug?**
  - A bug is the consequence/outcome of a coding fault

- **What is Defect?**
  - A **defect is a variation or deviation from the original business requirements**

- These two terms have very thin line of difference, In the Industry both are faults that need to be fixed.

- When a tester executes the test cases, he might come across the test result which is contradictory to expected result. **This variation in the test result is referred as a Software Defect.**

- These defects or variation are referred by different names in a different organization like **issues, problem, bug or incidents**.

Bug Repot

- While reporting the bug to developer, your Bug Report should contain the following information
- **Defect_ID** - Unique identification number for the defect.
- **Defect Description** - Detailed description of the Defect including information about the module in which Defect was found.
- **Version** - Version of the application in which defect was found.
- **Steps** - Detailed steps along with screenshots with which the developer can reproduce the defects.
- **Date Raised** - Date when the defect is raised

- **Reference-** where in you Provide reference to the documents like . requirements, design, architecture or maybe even screenshots of the error to help understand the defect.

- **Detected By -** Name/ID of the tester who raised the defect.

- **Status -** Status of the defect

- **Fixed by -** Name/ID of the developer who fixed it

- **Date Closed -** Date when the defect is closed

- **Severity** which describes the impact of the defect on the application

- **Priority** which is related to defect fixing urgency. Severity Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed respectively.

If the defect communication is done verbally, soon things become very complicated.

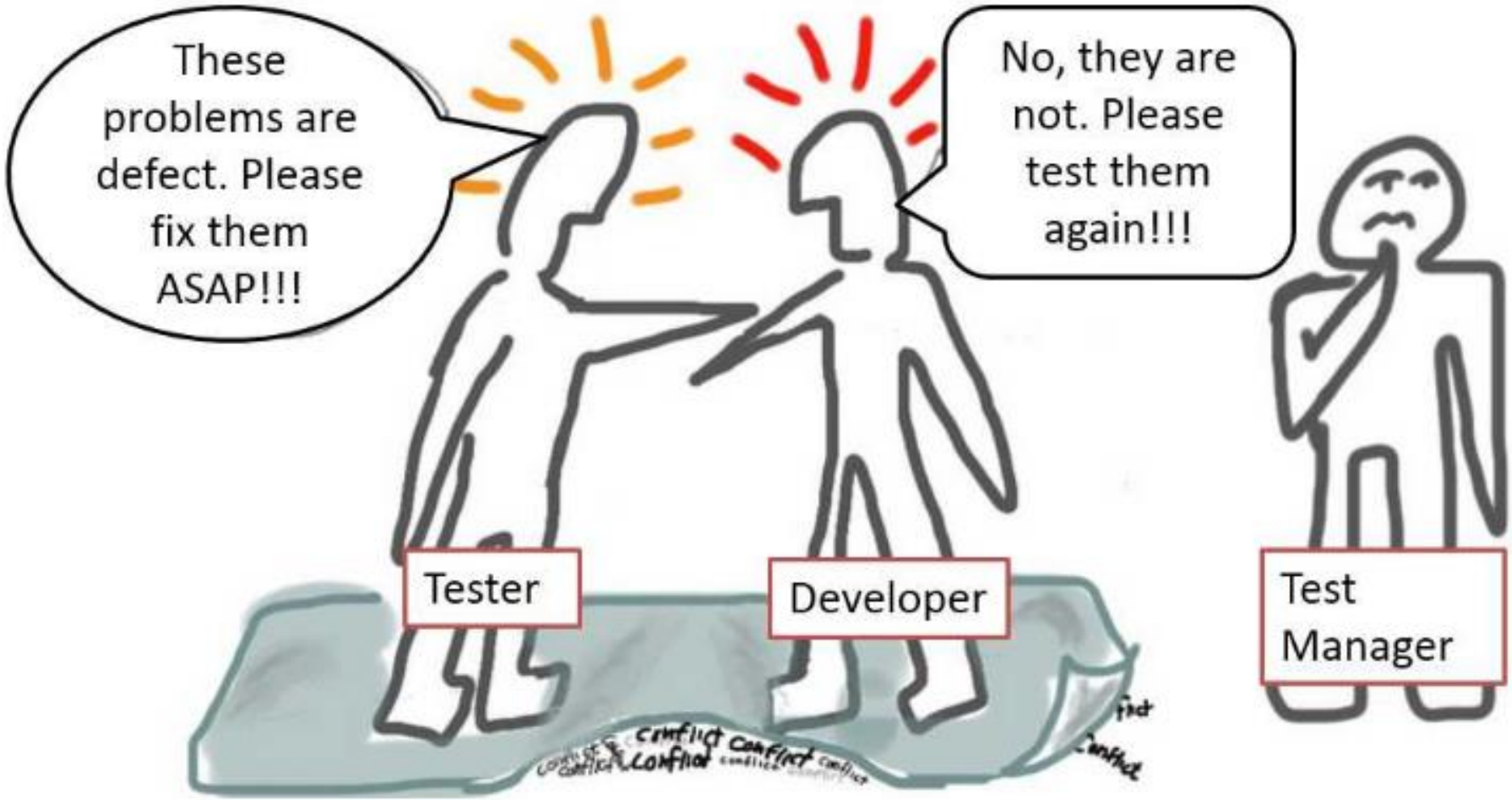
To control and effectively manage bugs you need a defect lifecycle.

Defect Management Process

Discovery

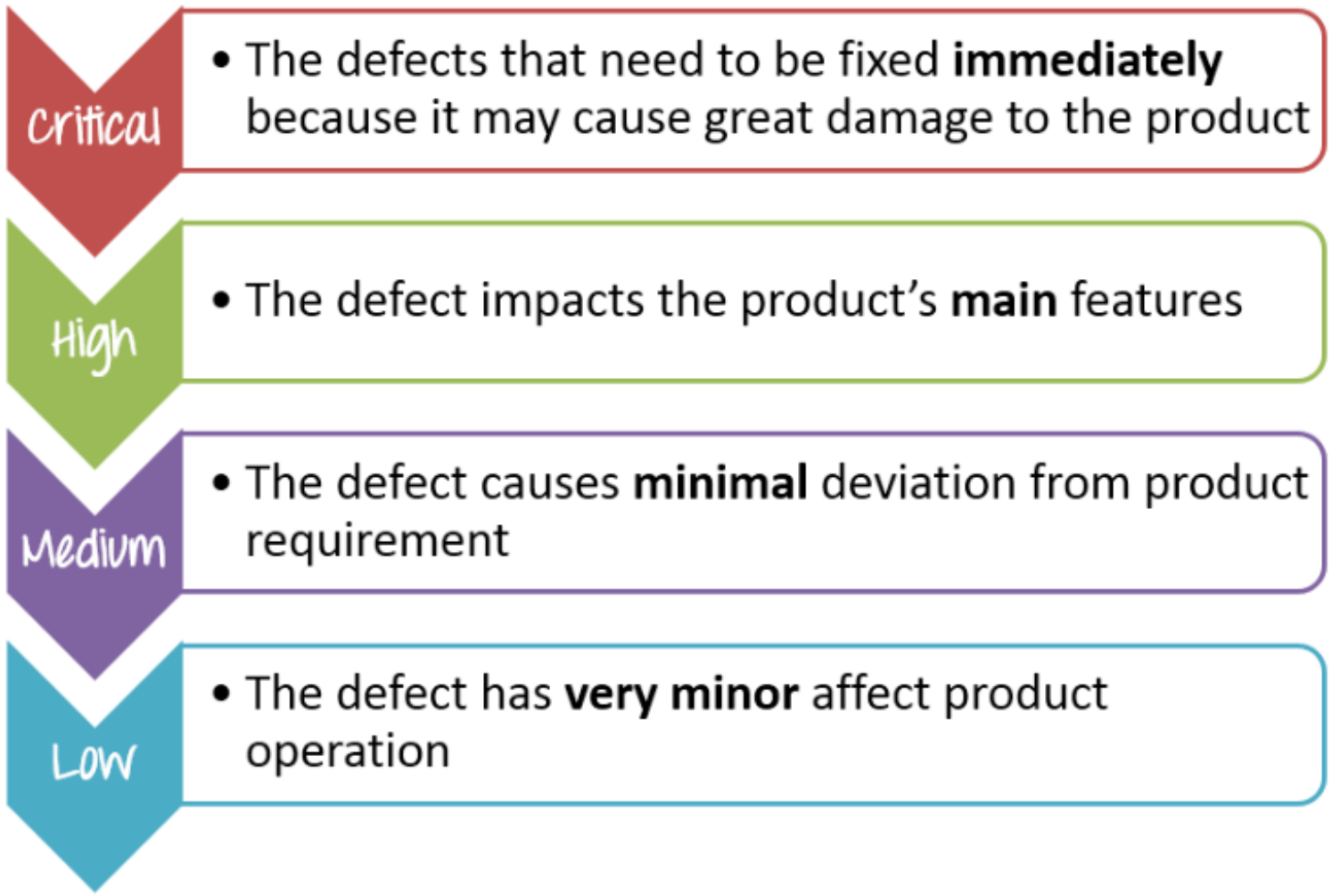
- In the discovery phase, the project teams have to discover as **many** defects as **possible,** before the end customer can discover it.

- A defect is said to be discovered and change to status **accepted** when it is acknowledged and accepted by the developer


These problems are defect. Please fix them ASAP!!!
No, they are not. Please test them again!!!
Tester
Developer
Test Manager
conflict

Categorization:

- Defect categorization help the software developers to prioritize their tasks.
- That means that this kind of priority helps the developers in fixing those defects first that are highly crucial



Critical | High | Medium | Low

1) The website performance is too slow

2) The login function of the website does not work properly

3) The GUI of the website does not display correctly on Mobile devices

4) The website could not remember the user login session

5) Some links doesn't work

No.	Description	Priority	Explanation
1	The website performance is too slow	High	The performance bug can cause huge inconvenience to user.
2	The login function of the website does not work properly	Critical	Login is one of the main function of the banking website if this feature does not work, it is serious bugs
3	The GUI of the website does not display correctly on mobile devices	Medium	The defect affects the user who use Smartphone to view the website.
4	The website could not remember the user login session	High	This is a serious issue since the user will be able to login but not be able to perform any further transactions
5	Some links doesn't work	Low	This is an easy fix for development guys and the user can still access the site without these links

Resolution

Once the defects are accepted and categorized, you can follow the following steps to fix the defect.

- **Assignment**: Assigned to a developer or other technician to fix, and changed the status to **Responding**.
- **Schedule fixing**: The developer side take charge in this phase. They will create a **schedule to fix these defects**, depend on the **defect priority.**
- **Fix the defect**: While the development team is fixing the defects, the Test Manager tracks the process of fixing defect compare to the above schedule.
- **Report the resolution**: Get a report of the resolution from developers when defects are fixed.

Verification

- After the development team **fixed** and **reported** the defect, the testing team **verifies** that the defects are actually resolved.
    - For example, in the above scenario, when the development team reported that they already fixed 61 defects, your team would test again to verify these defects were actually fixed or not.

Closure

- Once a defect has been resolved and verified, the defect is changed status as **closed**.

- If not, you have send a notice to the development to check the defect again.

Reporting

- The management board has right to know the defect status.
- They must understand the defect management process to support you in this project.
- Therefore, you must report them the current defect situation to get feedback from them.

Defect Metrics

- Out of 84 defects, we reported only 64 are actually defects.
- It means 20 defects are wrong. It is mistake in our testing
  - How can we measure the quality of test execution?

- **How to measure and evaluate the quality of the test execution?**

- There are 2 parameters which you can consider as following
  - Defect Rejection Ratio
  - Defect Leakage Ratio

- In the above scenario, you can calculate the
  - **defection rejection ratio** (DRR) is **20/84 = 0.238 (23.8 %).**

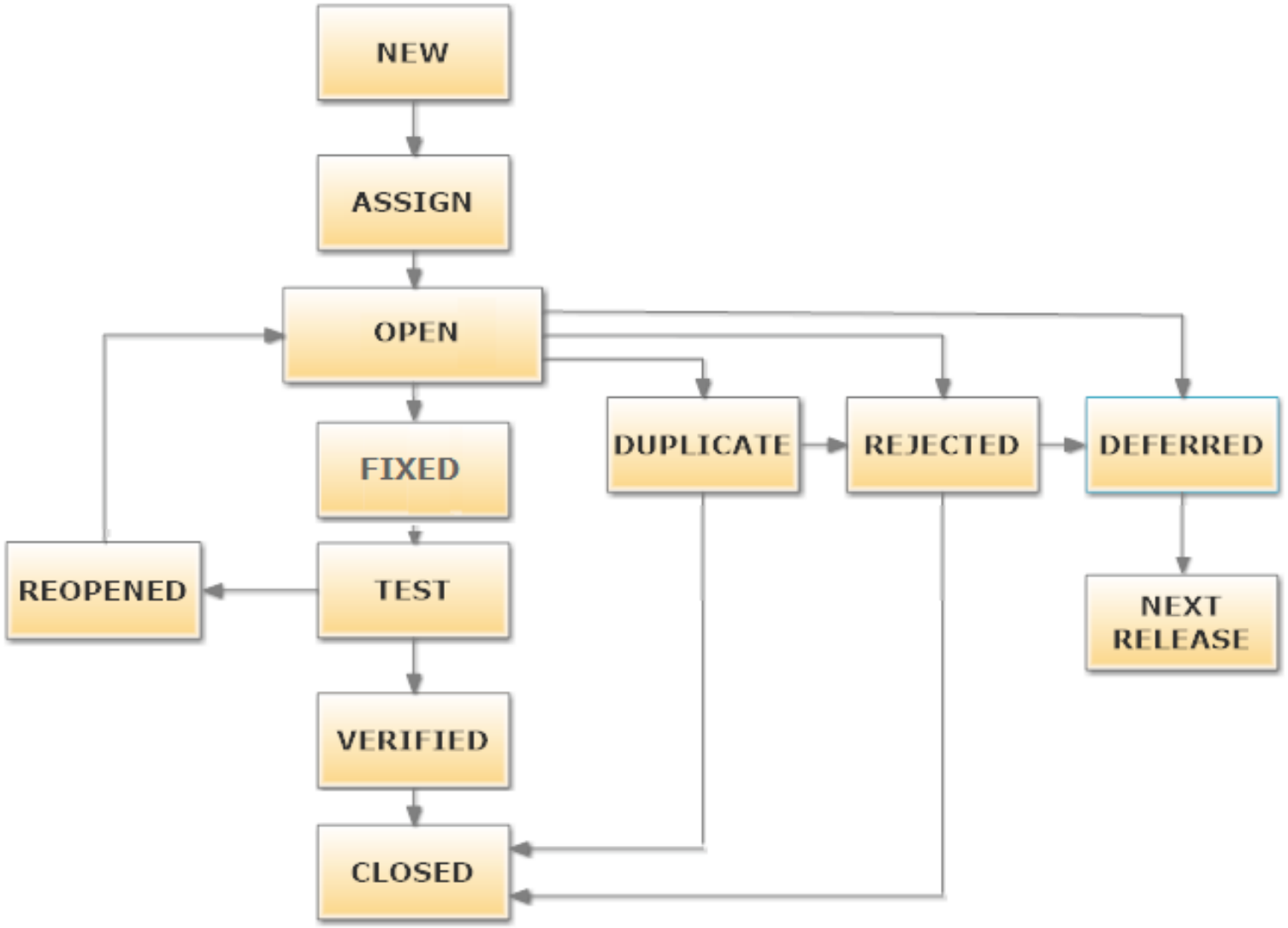
- Another example, supposed the Bank website has total **64** defects, but your testing team only detect **44** defects i.e. they missed **20** defects.
  - Therefore, you can calculate the defect leakage ratio (DLR) is 20/64 = **0.312** (31.2 %).

- **The smaller value of DRR and DLR**

  is the better quality of test execution is.

- What is the ratio range which is **acceptable**?
  - This range could be defined and accepted base in the project target **or you may refer the metrics of similar projects.**

Software Defect Life Cycle



- Defect or bug life cycle is a cycle through which a defect goes in its lifetime.
- Defect cycle starts when defect is found and it ends when the defect is closed

Status of a bug in the bug life cycle

- **New:**
  - When a tester finds a new defect.
  - He should provide a proper Defect document to the Development team to reproduce and fix the defect.
  - In this state, the status of the defect posted by tester is “New”

- **Assigned:**
  - Defects which are in the status of New will be **approved (if valid) and assigned to the development team** by Test Lead/Project Lead/Project Manager.
  - Once the defect is assigned then the status of the bug changes to “Assigned”

- **Open:**
  - The development team **starts analysing and works on the defect fix**

- **Fixed:**
  - When a developer makes the necessary code change and verifies the change,
  - then the status of the bug will be changed as **“Fixed”** and
  - the bug is **passed to the testing team**.

- **Test:**
  - If the status is “Test”, it means **the defect is fixed** and ready to do test whether it is fixed or not.

- **Verified:**
  - The tester **re-tests the bug after it got fixed by the developer**.
  - If there is no bug detected in the software, then the bug is fixed and the status assigned is “verified.”

- **Closed:**
  - **After verified the fix, if the bug is no longer exits then the status of bug will be assigned as “Closed.”**

- **Reopen:**
  - **If the defect remains same after the retest**, then the tester posts the defect using defect retesting document and changes the status to “Reopen”.
  - Again the bug goes through the life cycle to be fixed.

- **Duplicate:**
  - **If the defect is repeated twice** or the defect corresponds the same concept of the bug, the status is changed to “duplicate” by the development team.

- **Deferred:**
  - In some cases, Project Manager/Lead, may set the bug status as deferred.
    - If the bug found during end of release and the **bug is minor** or not important to fix immediately
    - If the bug is not related to current build
    - If it is expected to get fixed in the next release
    - Customer is thinking to change the requirement
    - In such cases the status will be changed as “deferred” and it will be fixed in the next release.

- **Rejected:**
  - If the system is working according to specifications and **bug is just due to some misinterpretation**
  - then Team lead or developers can mark such bugs as “Rejected”

- Some other statuses are:

- **Cannot be fixed:** Technology not supporting, Cost of fixing bug is more

- **Not Reproducible:** Platform mismatch, improper defect document, data mismatch, build mismatch, inconsistent defects.

- **Need more information:** In this case, the tester needs to add detailed reproducing steps and assign bug back to the development team for a fix. This won’t happen if the tester writes a good defect document.

GUI Testing

GUI Testing

- It is the process of **testing the system's Graphical User interface** of the application under test.

- GUI testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars-toolbars, menu bar, dialog boxes, etc

Checklist of GUI Testing

- Checks all the **GUI elements** for size, position, width, length and acceptance of characters or numbers
- Check you can **execute the intended functionality of the application** using the GUI
- Check **error messages are displayed correctly**
- Check **paging and soring in case of more records**.
- Check **readability of font used**.
- Check the **alignment of the text**
- Check that **the images have good clarity and properly aligned**.
- Check the **positioning of GUI elements** for different screen resolution.

How to Select Best Automation Testing Tool

How to select best tool for project?

- You want to support your test activities by means of a software tool, but **you don't know tools currently available in market..**

- **Which type of tool will best fit the requirement and the project budget**?

- Who on the **team has the skills to use the tool** once you have purchased it

- **The importance of the software testing tool selection: Less man power**
- **Type of test tools:**
  - **Open Source**
  - **Commercial:**
    - **Produced for sale** or to serve commercial purposes.
    - It has **more support and more features** from a vendor than open-source tools.
  - **Custom:**
    - In some Testing project, the testing environment, and the **testing process has special characteristics.** No open-source or commercial tool can meet the requirement. Therefore, the **Test Manager has to consider the development of the custom tool**.

Tool selection process

- **Step 1) Identify the requirement for tools**
  - **Precisely identify your test tool requirements**.
  - All the requirement must be **documented** and **reviewed** by project teams and the management board.

- **Step 2) Evaluate the tools and vendors**

  - **Test Manager should**
    - **Analyse the commercial and open source tools that are available in the market**, based on the project requirement.
    - Create a **tool shortlist** which best meets your criteria
    - One factor you should consider is **vendors**. You should consider the vendor's reputation, after sale support, tool update frequency, etc. while taking your decision.
    - Evaluate the quality of the tool by taking the **trial usage & launching a pilot**. Many vendors often make trial versions of their software available for download

- **Step 3) Estimate cost and benefit**
  - To ensure the test tool is beneficial for business, the Test Manager have to **balance** the following factors: **Value, Cost, Benefit**

- **Step 4) Make the final decision**
  - The Test Manager must have:
    - Have a **strong awareness** of the tool. It means you must understand which is the **strong** points and the **weak** points of the tool
    - **Balance** cost and benefit.

- Even with hours spent reading software manual and vendor information, you may still need to **try the tool in your actual working environment before buying the license**.

- **You should have the meeting with the project team, consultants to get the deeper knowledge of the tool.**

Test Management Tools

- Whether it is about capturing requirements, designing test cases, test execution reports, informing other team-members about testing progress etc... a test management tool is mandatory.

- Even small error in recording these details may lead to catastrophic effect and failure of the project.

- So, to manage all these details some test management tools can come very handy and useful.

Test Management Tools

- **Zephyr**
- HypTest
- PractiTest
- TestRail
- ReQtest
- TestPad
- Xray
- Test Collab
- TestMonitor