DEFECT & TEST MANAGEMENT

PMIT 6111: ST & QA

What is Defect?

- When expected result of test mismatched with actual result of the application, this mismatch is called **defect**.
- It can also be error, flaw, failure, or fault in a computer program. Most bugs arise from mistakes and errors made by developers, architects.
- Following are the common types of defects that occur during ***development***:
  - Arithmetic Defects
  - Logical Defects
  - Syntax Defects
  - Multithreading Defects
  - Interface Defects
  - Performance Defects

Defect Classification

- Defects are classified in two perspectives:
  - From the QA team perspective as **Priority**
  - From the development perspective as **Severity** (complexity of code to fix it).
- These are two major classifications that play an important role in the timeframe and the amount of work that goes in to fix defects.

Priority

- What is Priority?
  - Priority is defined as the order in which the defects should be resolved.
  - The priority status is usually set by the QA team while raising the defect against the dev team mentioning the timeframe to fix the defect.
  - The Priority status is set based on the requirements of the end users.
    - For example, if the company logo is incorrectly placed in the company's web page then the priority is high but it is of low severity.
- A Priority can be categorized in the following ways –
  - **Low** – This defect can be fixed after the critical ones are fixed.
  - **Medium** – The defect should be resolved in the subsequent builds.
  - **High** – The defect must be resolved immediately because the defect affects the application to a considerable extent and the relevant modules cannot be used until it is fixed.
  - **Urgent** – The defect must be resolved immediately because the defect affects the application or the product severely and the product cannot be used until it has been fixed.

Severity

- Severity is defined as the impishness of defect on the application and complexity of code to fix it from development perspective. It is related to the development aspect of the product.
- Severity can be decided based on how bad/crucial is the defect for the system.
- Severity status can give an idea about the deviation in the functionality due to the defect.
  - **Example** – For flight operating website, defect in generating the ticket number against reservation is high severity and also high priority.
- Severity can be categorized in the following ways –
  - **Critical /Severity 1** – Defect impacts most crucial functionality of Application and the QA team cannot continue with the validation of application under test without fixing it. For example, App/Product crash frequently.
  - **Major / Severity 2** – Defect impacts a functional module; the QA team cannot test that particular module but continue with the validation of other modules. For example, flight reservation is not working.
  - **Medium / Severity 3** – Defect has issue with single screen or related to a single function, but the system is still functioning. The defect here does not block any functionality. For example, Ticket# is a representation which does not follow proper alpha numeric characters like the first five characters and the last five as numeric.
  - **Low / Severity 4** – It does not impact the functionality. It may be a cosmetic defect, UI inconsistency for a field or a suggestion to improve the end user experience from the UI side. For example, the background colour of the Submit button does not match with that of the Save button.

What is Defect Logging and Tracking?

- ***Defect logging*****,** a process of finding defects in the application under test or product by testing or recording feedback from customers and making new versions of the product that fix the defects or the clients feedback.

- ***Defect tracking*** is an important process in software engineering as complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects gets multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.
  - **Examples - Hp Quality Center, IBM Rational Quality Manager**

Defect Tracking Parameters

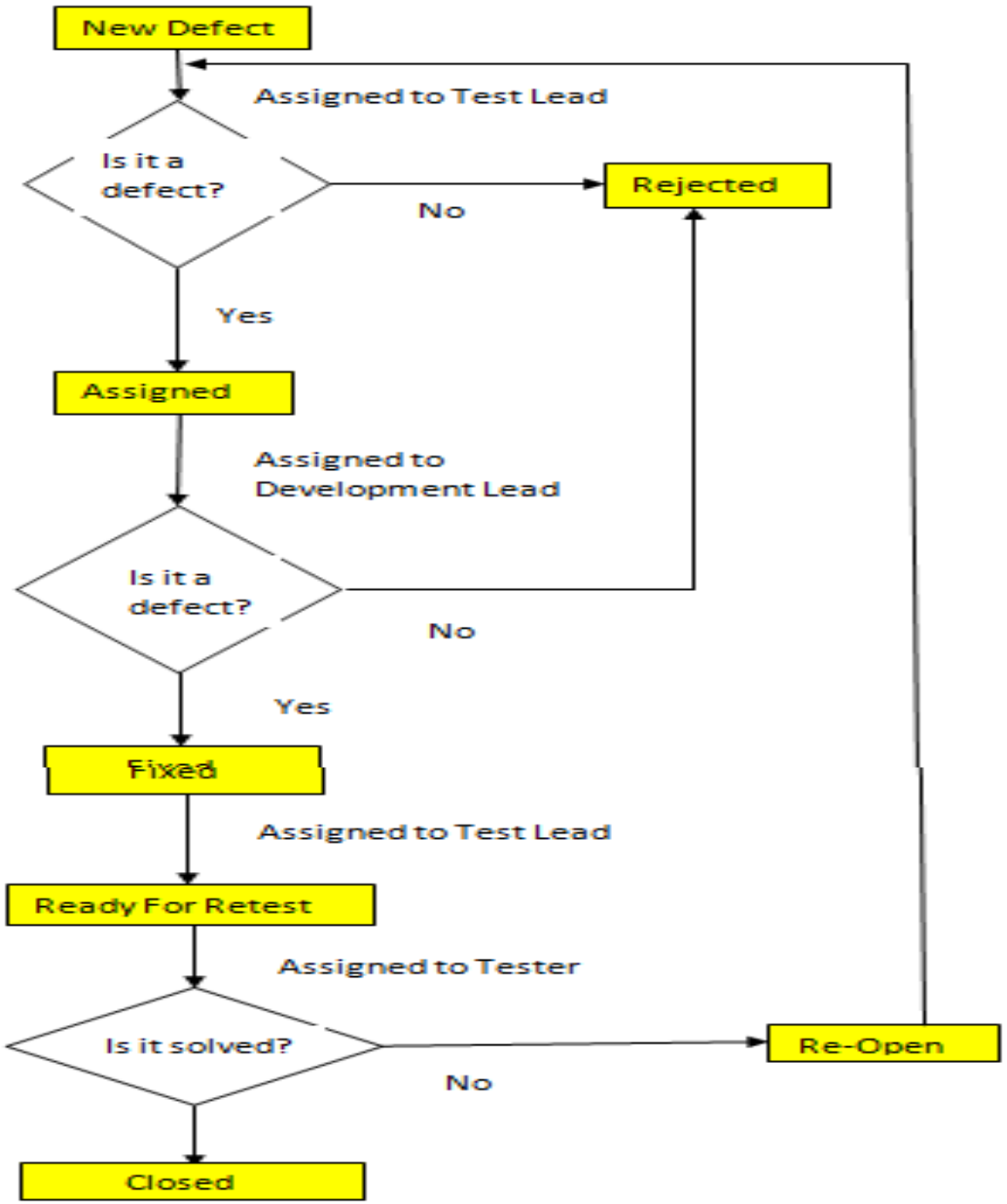
- Defects are tracked based on various parameters such as:
  - Defect Id
  - Priority
  - Severity
  - Created by
  - Created Date
  - Assigned to
  - Resolved Date
  - Resolved By
  - Status

Defect Tracking Process

- After a defect has been found it must be reported to the development team so that they can fix the issue.
- A tester logged a defect and the initial state of a defect is 'New'. If a business analyst is available within the project, then the tester assigns the defect to him. On absence of a ***business analyst*** the project lead or the coordinator reviews the defect. If the defect is valid then it will be assigned to the development team lead and the status will be 'Assigned' , otherwise the defect is marked as 'Rejected'. The BA or lead can marked the defect as 'Duplicate' if there was a same bug reported earlier and 'Deferred' if this is a future enhancement.
- Once the development team has fixed the issue and provides the test team a new build, the defect status is changed to 'Fixed' and it will be assigned back to the test lead. The test lead reviews it and assigns the defect back to the tester with the defect status as 'Ready For Retest'. The tester verifies it once again and if the functionality works as specified the tester marks the defect status as 'Closed' and also passes the relevant test case.
- On re-testing the defect if the issue still persists then the tester changes the status as 'Re-Opened' and assigns the defect back to the lead (the same cycle will be followed)

What is Defect Life Cycle?

- Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime.
- It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.


New Defect
Assigned to Test Lead
Is it a defect?
No
Rejected
Yes
Assigned
Assigned to Development Lead
Is it a defect?
No
Yes
Fixed
Assigned to Test Lead
Ready For Retest
Assigned to Tester
Is it solved?
No
Re-Open
Closed

Defect Life Cycle 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; viz - Deferred or Rejected.
- **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 Reporting

- The key to make a good report is providing the development team as much information as necessary to reproduce the bug. These can be broken into the following:
  - Give a brief description of the bug or defect
  - List the steps that are needed to reproduce the issue
  - Provide all the details on the test data used to get the issue
  - Provide all the required screen shots of the defect to the development team
  - Provide the opinion from a tester's perspective
- Generally the more information is shared by the test team, the easier it will be for the development team to determine the problem and fix the issue. But a bug report is a case against a product so the report should be concise such that someone who has never seen the system can follow the steps and reproduce the issue.

Test Management

- Test management is an essential part for Software Testing Life Cycle. To a great degree we can conclude that test management depends on proper estimation and judgment. Many external and internal factors impact the project as well as the project timeline and this is where the test estimation and risk analysis meet together. Estimation involves risk at all level and mainly the following factors influence test estimation:
  - Requirements
  - Available resources
  - Test Data
  - Fund available

Testing Estimation Techniques

- The following testing estimation techniques are proven to be accurate and are widely used –
  - PERT software testing estimation technique
  - UCP Method
  - WBS
  - Wideband Delphi technique
  - Function point/Testing point analysis
  - Percentage distribution
  - Experience-based testing estimation technique

PERT Software Testing Estimation Technique

- PERT software testing estimation technique is based on statistical methods in which each testing task is broken down into sub-tasks and then three types of estimation are done on each sub-tasks.
  - The formula used by this technique is –
    - **Test Estimate = (O + (4 × M) + E)/6**
  - Where,
    - **O** = Optimistic estimate (best case scenario in which nothing goes wrong and all conditions are optimal).
    - **M** = Most likely estimate (most likely duration and there may be some problem but most of the things will go right).
    - **L** = Pessimistic estimate (worst case scenario where everything goes wrong).
  - Standard Deviation for the technique is calculated as –
    - **Standard Deviation (SD) = (E – O)/6**

Use-Case Point Method

- **Step 1** – Count the no. of actors. Actors include positive, negative and exceptional.
- **Step 2** – Calculate unadjusted actor weights as
  - **Unadjusted Actor Weights = Total no. of Actors**
- **Step 3** – Count the number of use-cases.
- **Step 4** – Calculate unadjusted use-case weights as
  - **Unadjusted Use-Case Weights = Total no. of Use-Cases**
- **Step 5** – Calculate unadjusted use-case points as
  - **Unadjusted Use-Case Points = (Unadjusted Actor Weights + Unadjusted Use-Case Weights)**
- **Step 6** – Determine the technical/environmental factor (TEF). If unavailable, take it as 0.50.
- **Step 7** – Calculate adjusted use-case point as
  - **Adjusted Use-Case Point = Unadjusted Use-Case Points × [0.65 + (0.01 × TEF]**
- **Step 8** – Calculate total effort as
  - **Total Effort = Adjusted Use-Case Point × 2**

Work Breakdown Structure

- **Step 1** – Create WBS by breaking down the test project into small pieces.
- **Step 2** – Divide modules into sub-modules.
- **Step 3** Divide sub-modules further into functionalities.
- **Step 4** – Divide functionalities into sub-functionalities.
- **Step 5** – Review all the testing requirements to make sure they are added in WBS.
- **Step 6** – Figure out the number of tasks your team needs to complete.
- **Step 7** – Estimate the effort for each task.
- **Step 8** – Estimate the duration of each task.

Wideband Delphi Technique

- In Wideband Delphi Method, WBS is distributed to a team comprising of 3-7 members for re-estimating the tasks. The final estimate is the result of the summarized estimates based on the team consensus.
- This method speaks more on experience rather than any statistical formula. This method was popularized by Barry Boehm to emphasize on the group iteration to reach a consensus where the team visualized different aspects of the problems while estimating the test effort.

Function Point / Testing Point Analysis

- FPs indicate the functionality of software application from the user's perspective and is used as a technique to estimate the size of a software project.
- In testing, estimation is based on requirement specification document, or on a previously created prototype of the application. To calculate FP for a project, some major components are required. They are –
  - **Unadjusted Data Function Points** –
    - i) Internal Files, ii) External Interfaces
  - **Unadjusted Transaction Function Points** –
    - i) User Inputs, ii) User Outputs & iii) User Inquiries
  - **Capers Jones basic formula** –
    - Number of Test Cases = (Number of Function Points) × 1.2
  - **Total Actual Effort (TAE)** –
    - (Number of Test cases) × (Percentage of Development Effort /100)

Percentage Distribution

- In this technique, all the phases of Software Development Life Cycle (SDLC) are assigned effort in %. This can be based on past data from similar projects. For example –

Phase	% of Effort
Project Management	7%
Requirements	9%
Design	16%
Coding	26%
Testing (all Test Phases)	27%
Documentation	9%
Installation and Training	6%

All Testing Phases	% of Effort
Component Testing	16
Independent Testing	84
Total	**100**

Independent Testing	% of Effort
Integration Testing	24
System Testing	52
Acceptance Testing	24
Total	**100**

System Testing	% of Effort
Functional System Testing	65
Non-functional System Testing	35
Total	**100**

Test Planning and Design Architecture	50%
Review phase	50%

Experience-based Testing Estimation Technique

- This technique is based on analogies and experts. The technique assumes that you already tested similar applications in previous projects and collected metrics from those projects. You also collected metrics from previous tests. Take inputs from subject matter experts who know the application (as well as testing) very well and use the metrics you have collected and arrive at the testing effort.

Source & Reference

- https://www.tutorialspoint.com/software_testing_dictionary/
- https://www.tutorialspoint.com/estimation_techniques/estimation_techniques_testing.htm