CONTENTS

1. TEST CASE MANAGEMENT

Test Case Management is the process that allows an individual or a team of testers to store details on how testing will be done, plan testing activities, execute and report the status of execution. There are various tools that can be used for these activities, tools that have different approaches to testing and thus have different sets of features. Their principal functions are to maintain and plan manual testing, run or gather execution data from automated tests, manage multiple environments and to enter information about found defects. Many test management tools incorporate requirements management capabilities to streamline test case design from the requirements. Tracking of defects and project tasks are done within one application to further simplify the testing.

1.1 TEST CASE WORKFLOW

Depending on the various tools used for test case management, these will offer pre-defined or customizable test case workflows. Despite this, the below described are the most commonly used steps in a test case's lifecycle:

- New/Design – Tester has identified the scenario/requirement under test and has begun writing it
- Ready for Review – Tester has completed documenting the steps and other specific test requirements and the test case is ready to be reviewed by either a peer tester from inside the project or the project Business Analyst

- Approved – The test has been reviewed by the above mentioned and has been marked as approved or ready to be executed
- Automated – The test cases have been automated using various automation tools
- Obsolete – The test case no longer covers an active functionality or due to changes in the system is no longer a valid/executable scenario

1.2 ORGANIZING YOUR TEST CASES

One of the most challenging tasks in a project is keeping the test-suites maintainable over a long period of time. Especially if some window or workflow changes significantly the maintenance effort should be kept to a minimum.

It is also worth thinking about how to minimize the creation efforts of tests that contain a lot of similar or even the same steps. A typical use-case is launching the system under test or the login process or a very important and basic workflow like navigating to a certain part of the system under test.

Another aspect to think about is how to efficiently organize test-suites if different people work in your testing project.

The various test management tools used provide testing teams with means to structure and organize their test cases, however this does often is not enough.

Here are some criteria's that can be used when organizing your test cases from the high to low level:

- System under test (application) - the highest level when organizing, examples can be an online banking app, a flight booking app, a movie reservation app
- By release – depending on the delivery model, your application can have one or several incremental releases over a period of time until it reaches the final state
- By requirements – as the main starting point for all test cases, individual requirements also serve as a good way to way to group your test cases, for example in the movie app "The users should be able to choose their seats" all tests that are written for the seat reservation process will sit in the same location
- By module / application component – a more granular level than an actual requirement, for example the login module inside the banking application
- By test types - depending on the type of testing executed:
    - Sanity tests – the sanity test suite is usually run after each new version of code is applied in the testing environment and its main goal is to ensure the integrity of the application
    - Regression tests – the regression suite will ensure that the existing functionality is not affected by the introduction of new features
    - New functionality - these are the newest test cases that are added to confirm the new requirements
    - System Tests (End 2 End) – this is a type Software Testing that not only validates the software system under test but also checks its integration with external interfaces. Hence, the name "End-to-End". The purpose of End-to-End Testing is to exercise a complete production-like scenario.

1.3 EXECUTION & REPORTING

1.3.1 EXECUTION:

Manner of executing and test the actual system result against the expected result is test execution. Test execution can be done manually and by using automation suit. During the execution, the tester needs to make sure, that the user's need of the software is fulfilled. Test execution is conducted by referring the document created during test design as step by step process. The tester needs to keep the track of this step by step approach while executing the test cases.

Often during execution phase testers are faced with the issue that there isn't enough time to complete all the testing. Rather than finding a way to deal with this, most software test teams simply hope that they've done enough. Instead of that approach teams should consider prioritizing their software testing. Prioritizing your software testing will allow you to:

- Find the serious bugs sooner
- Deliver the results needed by the team faster
- Don't waste time on irrelevant tests

In an ideal world projects would have more testers and realistic deadlines but in the real world they often get little influence over those variables. They will be stuck with the resources they have and would need to make the most of them.

There isn't a precise way to prioritize each of your tests, as usually this requires knowledge of the product to do this. However, the following 6 points will help you:

1. Review defect trends from previous product releases
2. Identify areas where bugs have been fixed or new functionality added
3. Balance test priorities across early and later builds for the product
4. Make sure you have well defined meanings for each priority you use
5. Ensure you have consistent priorities across all your tests
6. Review test results from previous test runs

Basically the solution is not just to hope that you've done enough software testing. The solution is to find a system that enables you to priorities software tests and focus on what matters most.

1.3.2 REPORTING RESULTS

Test reporting gives the picture of test process and result for the particular testing cycle. To define the element in the test reporting the first thing that needs to be considered is whom the audiences of the test report are. For an example a project manager will like to see the high level picture of the testing, intermediate people will wish to view more detail and the client will expect the test reporting in the criteria such as requirement basis, feature basis.

Test report is prepared and communicated periodically like daily, weekly, month etc. This needs to be sent in different stages and time. In the future project result of test reports needs to be analyzed and apply the lesson learns.

Test reports usually contain elements such as

- application/requirement under test
- test execution status
- completed percentage
- plan vs. executed test cases
- test environment
- test execution by resources
- risk and mitigation if any
- defect summary
- test scenario and conditions

Often the projects will compile basic execution reports, this is generally prepared to be send to higher management from testing team to show the state of the test execution and test progress. Example of a test execution report:

Element	Description

Test item	Item, module or software that is been testing.
Scenario	Scenario of the test.
Responsible tester	Who tested the particular test item?
Date of test execution	Date of the test executed.
Status	Status of test execution.
Test environment	Detail of test environment. Detail of hardware, software and tools.
Risk and constrain	Any risk that impact the test execution and any constrain.
Test execution summary	This provide detail of test execution by software section, test type, defect rage etc.
Defect summary and detail	Defect count. Rate of open and closed defect. Severity and status wise defects.
Test cases detail	Planned and executed test cases. Status and result of execution and test case.
Test progress	Over all progress of test and which cycle the test execution is going on.

Other types of report can focus on more specific / detailed areas such as:

Test coverage report:

- Percentage completed
- Test scenario
- Software area
- Tested resource
- Tested date
- Test result

Defect summary report:

- Defect by severity
- Defects by priority
- Defects by assigned developer
- Defects by function
- Defects by software area
- Open and closed defects

1.4 TEST MANAGEMENT TOOLS – ZEPHYR INTRODUCTION

2. DEFECT MANAGEMENT

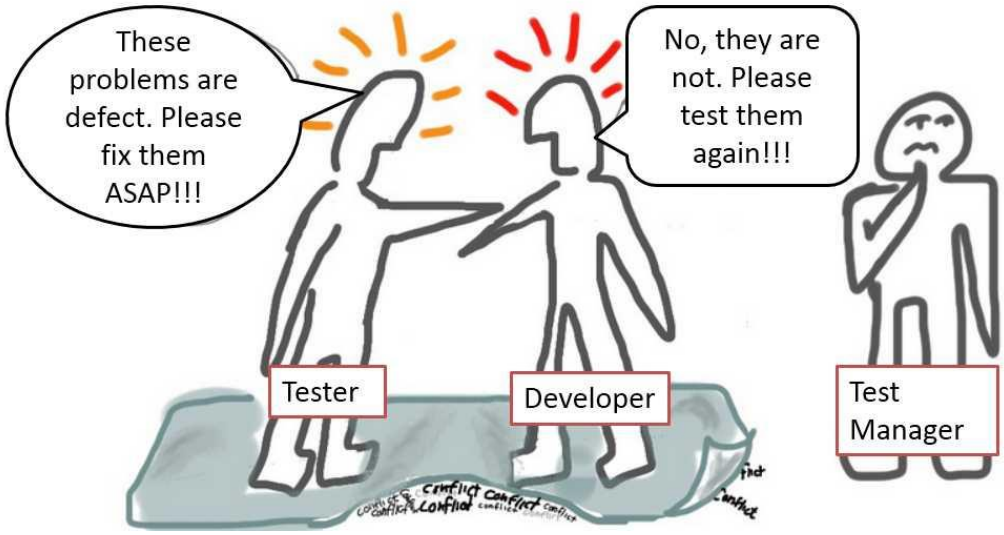
2.1. WHAT IS A DEFECT/BUG?

Jerry Weinberg states that “quality is value to some person”. Testing is one way to discover what that value is, and how it might be threatened by bugs. James Bach defines the term software bug as “anything that threatens quality”. Putting these two definitions together extends the definition of a defect to “anything that threatens the value of the software to some person.”

A **defect is an error or a bug**, in the application which is created. A programmer while designing and building the software can make mistakes or error. These mistakes or errors mean that there are flaws in the software. These are called defects.

- When actual result deviates from the expected result while testing a software application or product then it results into a defect. Hence, any deviation from the specification mentioned in the product functional specification document is a defect. In different organizations it’s called differently like bug, issue, incidents or problem.
- When the result of the software application or product does not meet with the end user expectations or the software requirements then it results into a Bug or Defect. These defects or bugs occur because of an error in logic or in coding which results into the failure or unpredicted or unanticipated results.

A defect can be introduced in any phase of Software Development Life Cycle (SDLC) so it is very important that test team is involved from beginning of SDLC for detecting and removal of defects. The earliest the defect is detected and rectified, the minimal cost of quality will incur.



2.2. DEFECT DISCOVERY & CATEGORIZATION

Software defects usually arise because of the following reasons:

- The person using the software application or product may not have enough knowledge of the product.
- Maybe the software is used in the wrong way which leads to the defects
- The developers may have coded incorrectly and there can be defects present in the design.
- Incorrect setup of the testing environments.

While testing when a tester executes the test cases, he might observe that the actual test results do not match from the expected results. The variation in the expected and actual results is known as incident which will require investigation.

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.
- Expected vs Actual Results – This is often a best practice that is usually included together with the Steps. It offers the developer an exact insight in what should happen vs what is actually happening in the application. What should happen is usually taken from the acceptance criteria
- 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, more on this later
- 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.

2.2.1 SEVERITY & PRIORITY

The severity of the defect shows how sever the defect is in terms of damaging to other systems, businesses, environment and lives of people, depending on the nature of the application system. The severities are normally ranked and categorized in 4 or 5 levels, depending on organization's definition.

- Blocker: This means the defect is a show stopper with high potential damages and has no workaround to avoid the defect. An example could be the application does not launch at all and causes the operating system to shut down. This requires immediate attention and action and fix.
- Critical: This means that some major functionalities of the applications are either missing or do not work and there is no workaround.
- Major: This means that some major functionality do not work, but, a workaround exists to be used as a temporary solution.
- Medium: This means that the failure causes inconvenience and annoyance.
- Minor: This is not normally a defect and a suggestion to improve a functionality. This can be GUI or viewing preferences.

Once the severity is determined, next is to see how to prioritize the resolution. The priority determines how quickly the defect should be fixed. The priority normally concerns the business importance such as impact on the project and the likely success of the product in the marketplace. Like severity, priority is also categorized in to 4 or 5 levels.

- P1 – Urgent: Means extremely urgent and requires immediate resolution
- P2 – High: Resolution requirement for next external release
- P3 – Medium: Resolution required for the first deployment (rather than all deployments)
- P4 – Low: Resolution desired for the first deployment or subsequent future releases

Examples

1. Low Severity, Low Priority

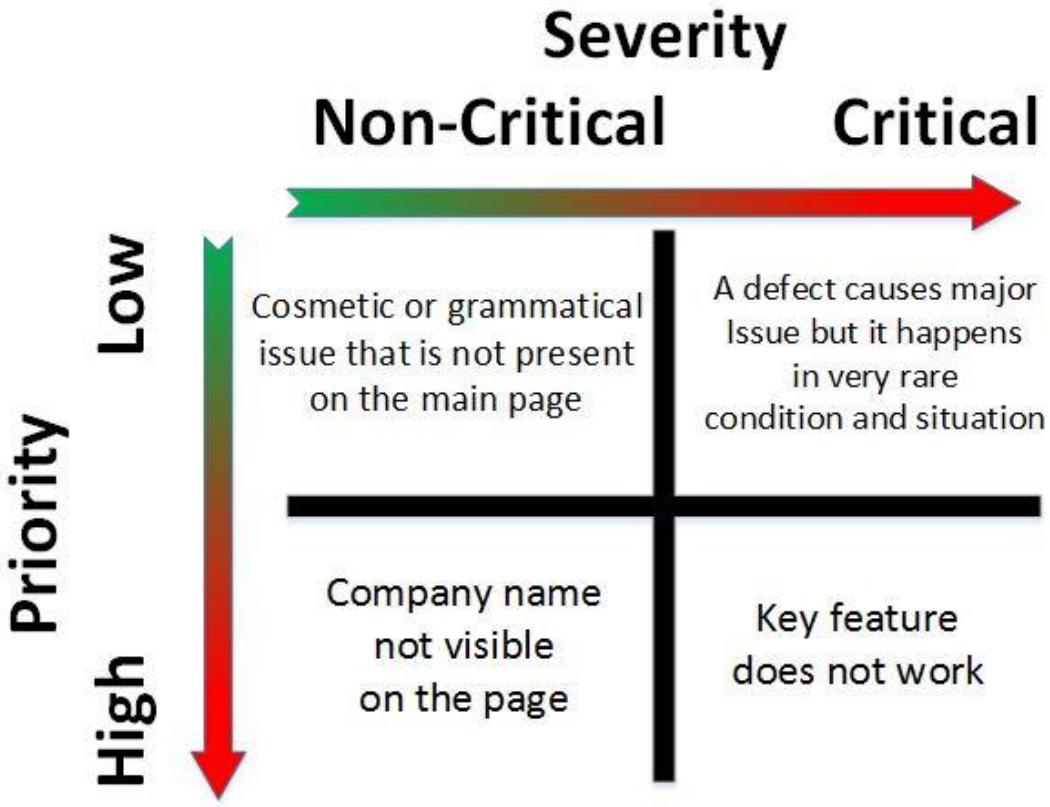
Suppose an application (web) is made up of 20 pages. On one of the pages out of the 20 which is visited very infrequently, there is a sentence with a grammatical error. Even though it's a mistake

on this website, users can understand its meaning without any difficulty. This bug may go unnoticed to the eyes of many and won't affect any functionality or the credibility of the company.

2. Low Severity, High Priority
   - While developing a site for Pepsi, by mistake a logo sign of coke is embedded. This does not affect functionality in any way but has high priority to be fixed.
   - Any typo mistakes or glaring spelling mistakes on home page

3. High Severity, Low Priority
   - Incase application works perfectly for 50000 sessions but beings to crash after higher number of sessions. This problem needs to be fixed but not immediately.
   - Any report generation not getting completed 100% - Means missing Title, Title Columns but having proper data enlisted. We could have this fixed in the next build but missing report columns is a High Severity defect.

4. High Severity, High Priority
   - Now assume a windows-based application, a word-processor let's say. As you open any file to be viewed it in, it crashes. Now, you can only create new files but as you open them, the word-processor crashes. This completely eliminates the usability of that word-processor as you can't come back and edit your work on it, and also affects one of the major functionalities of the application. Thus, it's a severe bug and should be fixed immediately.
   - Let's say, as soon as the user clicks login button on Gmail site, some junk data is displayed on a blank page. Users can access the gmail.com website, but are not able to login successfully and no relevant error message is displayed. This is a severe bug and needs topmost priority.



2.3. DEFECT LIFECYCLE

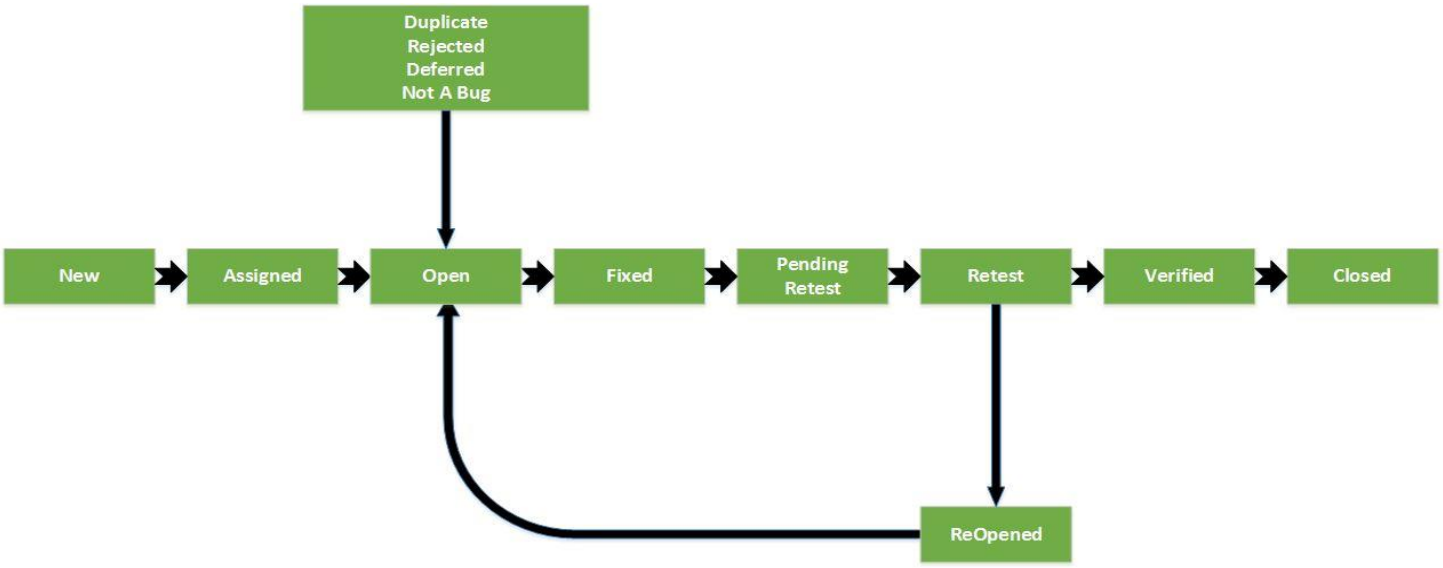
Defect Life Cycle is the journey of a defect from its identification to its closure. The Life Cycle varies from organization to organization and is governed by the software testing process the organization or project follows and/or the Defect tracking tool being used.

- **New:** When a new defect is logged and posted for the first time. It is assigned a status NEW.
- **Assigned:** Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to developer team
- **Open**: The developer starts analyzing and works on the defect fix
- **Fixed**: When developer makes necessary code change and verifies the change, he or she can make bug status as "Fixed."
- **Pending retest**: Once the defect is fixed the developer gives particular code for retesting the code to the tester. Since the testing remains pending from the testers end, the status assigned is "pending request."
- **Retest**: Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and change the status to "Re-test."

- 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."
- Reopen: If the bug persists even after the developer has fixed the bug, the tester changes the status to "reopened". Once again the bug goes through the life cycle.
- Closed: If the bug is no longer exists then tester assigns the status "Closed."

Often when closing an issue some it is needed to mark a certain Resolution for the specified defect, this is done by sometimes a separate field in the various tools or via comment when closed. Here are some examples of Resolutions:

- Duplicate: If the defect is repeated twice or the defect corresponds the same concept of the bug, the status is changed to "duplicate."
- Rejected: If the developer feels the defect is not a genuine defect then it changes the defect to "rejected."
- Deferred: If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status "Deferred" is assigned to such bugs
- Not a bug: If it does not affect the functionality of the application then the status assigned to a bug is "Not a bug".



2.4.JIRA OVERVIEW

What is JIRA?

JIRA is one of the most used defect tracking/project management tools used in software development projects. JIRA is a web based, open source and licensed product developed and owned by Atlassian, Inc

What can JIRA do?

- JIRA lets you prioritize, assign, track, report and audit your 'issues,' whatever they may be — from software bugs and help-desk tickets to project tasks and change requests.
- Customizable reporting allows you to monitor the progress of your issues with detailed graphs and charts.
- Map your business process with a custom workflow
- Integrate JIRA with other systems with their open API and 100+ free plugins.

Live presentation of JIRA and tutorial on ACAD Test JIRA

1. JIRA Issues and Issue types
    a. What is a JIRA Issue
    b. Issue Types
2. Issue Attributes

Statuses: Different statuses are used to indicate the progress of a project like To do, InProgress, Open, Closed, ReOpened, and Resolved.
Likewise, you have resolutions and priorities, in resolution it again tells about the progress of issue like Fixed, Won't fix, Duplicate, Incomplete, Cannot reproduce, Done
Set the priorities of the issue whether an issue is critical, major, minor, blocker and Trivial.

3. How to create issues & sub tasks in JIRA
4. Searching for Issues in JIRA
5. JIRA workflows

Source: http://www.guru99.com/jira-tutorial-a-complete-guide-for-beginners.html#1

Other introduction: https://www.youtube.com/watch?time_continue=28&v=xrCJv0fTyR8