K.T.S.P.Madal's

# Hutatma Rajguru Mahavidyalaya, Rajgurunagar

Tal-Khed, Dist.-Pune 410505.

## TY.BSc (Computer Science)

## Semester-VI

## Subject- Software Testing Tools

According to new CBCS syllabus w.e.f.2019-2020

Prof.S.V.Patole

Department of Computer Science

Hutatma Rajguru Mahavidyalaya, Rajgurunagar.

# Unit 4: Defect Report

## What is defect?

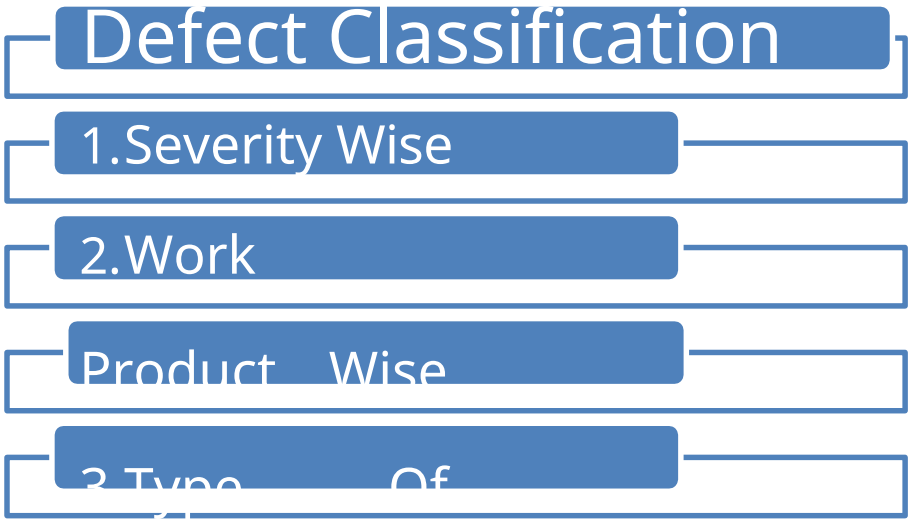
**Defect:** A **defect is an error or a bug**, in the application which is created. A programmer while designing and building the software can make mistakes or errors. These mistakes or errors mean that there are flaws in the software. These are called defects.

Defects in any system may arise in **various stages of development life cycle.** At each stage, the impact and cost of fixing defects are dependent on various aspects including the defect arising stage.

## Different causes of software defects

- Miscommunication of requirements introduces error in code
- Unrealistic time schedule for development
- Lack of designing experience
- Lack of coding practices experience
- Human factors introduces errors in code
- Lack of version control
- Buggy third-party tools
- Last minute changes in the requirement introduce error
- Poor Software testing skill

## Defect Classification



## Severity Wise:

- Major: A defect, which will cause an observable product failure or departure from requirements.
- Minor: A defect that will not cause a failure in execution of the product.
- Fatal: A defect that will cause the system to crash or close abruptly or effect other applications.

## Work product wise:

- **SSD:** A defect from System Study document
- **FSD:** A defect from Functional Specification document
- **ADS:** A defect from Architectural Design Document
- **DDS:** A defect from Detailed Design document
- **Source code:** A defect from Source code
- **Test Plan/ Test Cases:** A defect from Test Plan/ Test Cases
- **User Documentation:** A defect from User manuals, Operating manuals

## Type of Errors Wise:

- **Comments:** Inadequate/ incorrect/ misleading or missing comments in the source code
- **Computational Error:** Improper computation of the formulae / improper business validations in code.
- **Data error:** Incorrect data population / update in database
- **Database Error:** Error in the database schema/Design
- **Missing Design:** Design features/approach missed/not documented in the design document and hence does not correspond to requirements
- **Inadequate or sub optimal Design:** Design features/approach needs additional inputs for it to be complete Design features described does not provide the best approach (optimal approach) towards the solution required
- **In correct Design:** Wrong or inaccurate Design
- **Ambiguous Design:** Design feature/approach is not clear to the reviewer. Also includes ambiguous use of words or unclear design features.
- **Boundary Conditions Neglected:** Boundary conditions not addressed/incorrect
- **Interface Error:** Internal or external to application interfacing error, Incorrect handling of passing parameters, Incorrect alignment, incorrect/misplaced fields/objects, un friendly window/screen positions
- **Logic Error:** Missing or Inadequate or irrelevant or ambiguous functionality in source code

- **Message Error:** Inadequate/ incorrect/ misleading or missing error messages in source code
- **Navigation Error:** Navigation not coded correctly in source code
- **Performance Error:** An error related to performance/optimality of the code
- **Missing Requirements:** Implicit/Explicit requirements are missed/not documented during requirement phase
- **Inadequate Requirements:** Requirement needs additional inputs for to be complete
- **Incorrect Requirements:** Wrong or inaccurate requirements
- **Ambiguous Requirements:** Requirement is not clear to the reviewer. Also includes ambiguous use of words – e.g. Like, such as, may be, could be, might etc.
- **Sequencing / Timing Error:** Error due to incorrect/missing consideration to timeouts and improper/missing sequencing in source code.
- **Standards:** Standards not followed like improper exception handling, use of E & D Formats and project related design/requirements/coding standards
- **System Error:** Hardware and Operating System related error, Memory leak
- **Test Plan / Cases Error:** Inadequate/ incorrect/ ambiguous or duplicate or missing - Test Plan/ Test Cases & Test Scripts, Incorrect/Incomplete test setup
- **Typographical Error:** Spelling / Grammar mistake in documents/source code
- **Variable Declaration Error:** Improper declaration / usage of variables, Type mismatch error in source code

## Status Wise:

- Open
- Closed
- Deferred
- Cancelled

## Defect Management Process

The process of finding defects and reducing them at the lowest cost is called as Defect Management Process.


**Defect Prevention** -- Implementation of techniques, methodology and standard processes to reduce the risk of defects.

**Deliverable Baseline** -- Establishment of milestones where deliverables will be considered complete and ready for further development work. When a deliverable is base lined, any further changes are controlled. Errors in a deliverable are not considered defects until after the deliverable is base lined.

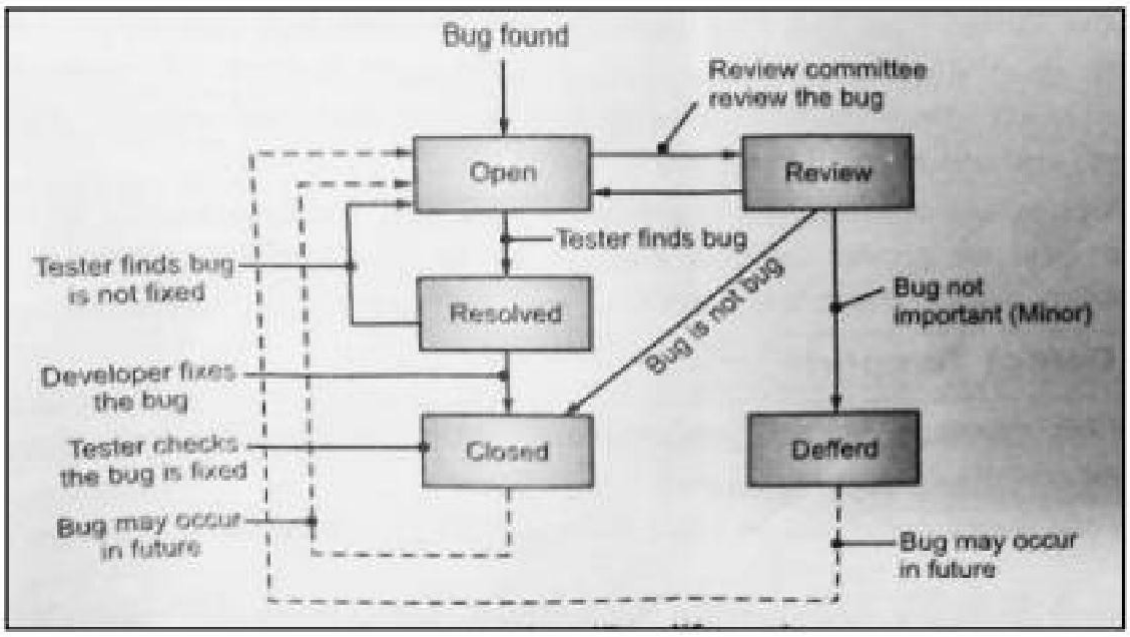
**Defect Discovery** -- Identification and reporting of defects for development team acknowledgment. A defect is only termed discovered when it has been documented and acknowledged as a valid defect by the development team member(s) responsible for the component(s) in error.

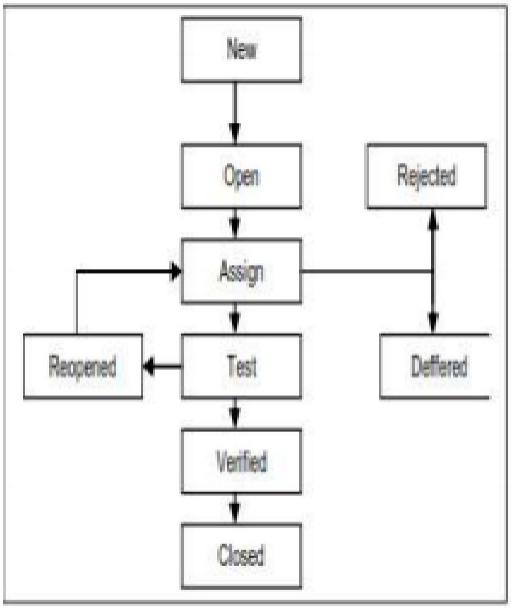
**Defect Resolution --** Work by the development team to prioritize, schedule and fix a defect, and document the resolution. This also includes notification back to the tester to ensure that the resolution is verified.

**Process Improvement** -- Identification and analysis of the process in which a defect originated to identify ways to improve the process to prevent future occurrences of similar defects. Also the validation process that should have identified the defect earlier is analyzed to determine ways to strengthen that process.

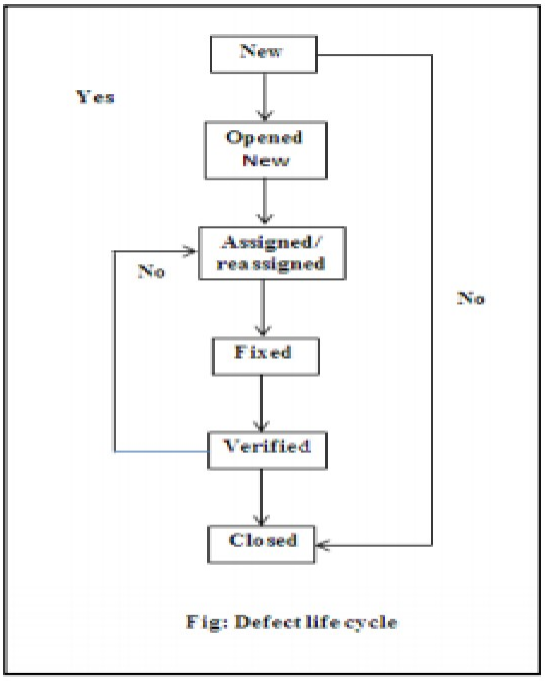
**Management Reporting --** Analysis and reporting of defect information to assist management with risk management, process improvement and project management.

## Defect Life Cycle




OR




OR

DEFECT LIFE CYCLE (Bug 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 the bug is posted for the first time, its state will be “NEW”. This means that the bug is not yet approved.
- **Open**: After a tester has posted a bug, the lead of the tester approves that the bug is genuine and he changes the state as “OPEN”.
- **Assign:** Once the lead changes the state as “OPEN”, he assigns the bug corresponding developer or developer team. The state of the bug now is changed to “ASSIGN”.
- **Test/Retest**: Once the developer fixes the bug, he has to assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to “TEST”. It specifies that the bug has been fixed and is released to testing team.// At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.
- **Deferred**: The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.
- **Rejected**: If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to “REJECTED”.
- **Verified:** Once the bug is fixed and the status is changed to “TEST”, the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to “VERIFIED”.
- **Reopened:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “REOPENED”. The bug traverses the life cycle once again.
- **Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “CLOSED”. This state means that the bug is fixed, tested and approved.
- **Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as „Fixed“ and the bug is passed to testing team.
- **Pending retest:** After fixing the defect the developer has given that particular code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest.

## Defect Prevention Process

- **“**Prevention is better than cure” applies to defects in the software development life cycle
- Defects, as defined by software developers, are variances from a desired attribute. These attributes include complete and correct requirements and specifications as drawn from the desires of potential customers.

- Thus, defects cause software to fail to meet requirements and make customers unhappy.
- when a defect gets through during the development process, the earlier it is diagnosed, the easier and cheaper is the rectification of the defect.
- The end result in prevention or early detection is a product with zero or minimal defects.


Identify Critical Risks -- Identify the critical risks facing the project or system. These are the types of defects that could jeopardize the successful construction, delivery and/or operation of the system.

Estimate Expected Impact -- For each critical risk, make an assessment of the financial impact if the risk becomes a problem.

Minimize Expected Impact -- Once the most important risks are identified try to eliminate each risk. For risks that cannot be eliminated, reduce the probability that the risk will become a problem and the financial impact should that happen.

The five general activities of defect prevention are:

1. **Software Requirements Analysis**

Defects introduced during the requirements and design phase are not only more probable but also are more severe and more difficult to remove.

Front-end errors in requirements and design cannot be found and removed via testing, but instead need pre-test reviews and inspections.

2. **Reviews: Self-Review and Peer Review**

Self-review is one of the most effective activity in uncovering the defects which may later be discovered by a testing team or directly by a customer.

A self-review of the code helps reduce the defects related to algorithm mplementations, incorrect logic or certain missing conditions.

Peer review is similar to self-review in terms of the objective – the only difference is that it is a peer (someone who understands the functionality of the code very well) who reviews the code

### 3. Defect Logging and Documentation

Effective defect tracking begins with a systematic process. A structured tracking process begins with initially logging the defects, investigating the defects, then providing the structure to resolve them. Defect analysis and reporting offer a powerful means to manage defects and defect depletion trends, hence, costs.

**Root Cause Analysis and Preventive Measures Determination**

After defects are logged and documented, the next step is to analyze them

**Reducing the defects to improve the quality:** The analysis should lead to implementing changes in processes that help prevent defects and ensure their early detection.

**Applying local expertise:** The people who really understand what went wrong are the people present when the defects were inserted – members of the software engineering team. They can give the best suggestions for how to avoid such defects in the future.

**Targeting the systematic errors:** There may be many errors or defects to be handled in such an analysis forum; however, some mistakes tend to be repeated. These systematic errors account for a large portion of the defects found in the typical software project.

### 5. Embedding Procedures into Software Development Process

Implementation is the toughest of all activities of defect prevention.

It requires total commitment from the development team and management.

A plan of action is made for deployment of the modification of the existing processes or introduction of the new ones with the consent of management and the team.

## Defect Report Template

- A defect report documents an anomaly discovered during testing.
- It includes all the information needed to reproduce the problem, including the author, release/build number, open/close dates, problem area, problem description, test environment, defect type, how it was detected, who detected it, priority, severity, status, etc. After uncovering a defect (bug), testers generate a formal defect report.
- The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it.

| | |
|---|---|
| **ID** | Unique identifier given to the defect. (Usually Automated) |
| **Project** | Project name. |
| **Product** | Product name. |
| **Release Version** | Release version of the product. (e.g. 1.2.3) |
| **Module** | Specific module of the product where the defect was detected. |
| **Detected Build Version** | Build version of the product where the defect was detected (e.g. 1.2.3.5) |
| **Summary** | Summary of the defect. Keep this clear and concise. |
| **Description** | Detailed description of the defect. Describe as much as possible but without repeating anything or using complex words. Keep it simple but comprehensive. |
| **Steps to Replicate** | Step by step description of the way to reproduce the defect. Number the steps. |
| **Actual Result** | The actual result you received when you followed the steps. |
| **Expected Results** | The expected results. |
| **Attachments** | Attach any additional information like screenshots and logs. |
| **Remarks** | Any additional comments on the defect. |
| **Defect Severity** | Severity of the Defect. |
| **Defect Priority** | Priority of the Defect. |
| **Reported By** | The name of the person who reported the defect. |
| **Assigned To** | The name of the person that is assigned to analyze/fix the defect. |
| **Status** | The status of the defect. |
| **Fixed Build Version** | Build version of the product where the defect was fixed (e.g. 1.2.3.9) |

## Estimate Expected Impact Of A Defect

- **Defect Impact:** The degree of severity that a defect has on the development or operation of a component or system.

- **How to Estimate the defect impact**

1. Once the critical risks are identified, the financial impact of each risk should be estimated.

2. This can be done by assessing the impact, in dollars, if the risk does become a problem combined with the probability that the risk will become a problem.

3. The product of these two numbers is the expected impact of the risk.

4. The expected impact of a risk (E) is calculated

   as E = P * I, where:

   P= probability of the risk becoming a problem and

   I= Impact in dollars if the risk becomes a problem.

Once the expected impact of each risk is identified, the risks should be prioritized by the expected impact and the degree to which the expected impact can be reduced. While guess work will constitute a major role in producing these numbers, precision is not important. What will be important is to identify the risk, and determine the risk's order of magnitude. Large, complex systems will have many critical risks. Whatever can be done to reduce the probability of each individual critical risk becoming a problem to a very small number should be done. Doing this increases the probability of a successful project by increasing the probability that none of the critical risks will become a problem.

One should assume that an individual critical risk has a low probability of becoming a problem only when there is specific knowledge justifying why it is low. For example, the likelihood that an important requirement was missed may be high if developers have not involved users in the project. If users have actively participated in the requirements definition, and the new system is not a radical departure from an existing system or process, the likelihood may be low.

For example:

o An organization with a project of 2,500 function points and was about medium at defect discovery and removal would have 1,650 defects remaining after all defect removal and discovery activities.

o The calculation is 2,500 x 1.2 = 3,000 potential defects.

o The organization would be able to remove about 45% of the defects or 1,350 defects.

o The total potential defects (3,000) less the removed defects (1,350) equals the remaining defects of 1,650.

## Techniques For Finding Defects

- Defects are found either by preplanned activities specifically intended to uncover defects (e.g., quality control activities such as inspections, testing, etc.) or by accident (e.g., users in production).
  Techniques to find defects can be divided into three categories:

- ***Static techniques***: Testing that is done without physically executing a program or system. A code review is an example of a static testing technique.

- ***Dynamic techniques***: Testing in which system components are physically executed to identify defects. Execution of test cases is an example of a dynamic testing technique.

- ***Operational techniques***: An operational system produces a deliverable containing a defect found by users, customers, or control personnel -- i.e., the defect is found as a result of a failure.

- While it is beyond the scope of this study to compare and contrast the various static, dynamic, and operational techniques, the research did arrive at the following conclusions:

- Both static and dynamic techniques are required for an effective defect management program. In each category, the more formally the techniques were integrated into the development process, the more effective they were.
  Since static techniques will generally find defects earlier in the process, they are more efficient at finding defects.

## Reporting a Defect

- **Be specific:**

  1. Specify the exact action: Do not say something like 'Select ButtonB'. Do you mean 'Click ButtonB' or 'Press ALT+B' or 'Focus on ButtonB and click ENTER'? Of course, if the defect can be arrived at by using all the three ways, it's okay to use a generic term as 'Select' but bear in mind that you might just get the fix for the 'Click ButtonB' scenario. [Note: This might be a highly unlikely example but it is hoped that the message is clear.]
  2. In case of multiple paths, mention the exact path you followed: Do not say something like "If you do 'A and X' or 'B and Y' or 'C and Z', you get D." Understanding all the paths at once will be difficult. Instead, say "Do 'A and X' and you get D." You can, of course, mention elsewhere in the report that "D can also be got if you do 'B and Y' or 'C and Z'."
  3. Do not use vague pronouns: Do not say something like "In ApplicationA, open X, Y, and Z, and then close it." What does the 'it' stand for? 'Z' or, 'Y', or 'X' or 'ApplicationA'?"

- **Be detailed:**

  1. Provide more information (not less). In other words, do not be lazy. Developers may or may not use all the information you provide but they sure do not want to beg you for any information you have missed.

- **Be objective:**

  1. Do not make subjective statements like "This is a lousy application" or

“You fixed it real bad.”

2. Stick to the facts and avoid the emotions.

- **Reproduce the defect:**

    1. Do not be impatient and file a defect report as soon as you uncover a defect. Replicate it at least once more to be sure. (If you cannot replicate it again, try recalling the exact test condition and keep trying. However, if you cannot replicate it again after many trials, finally submit the report for further investigation, stating that you are unable to reproduce the defect anymore and providing any evidence of the defect if you had gathered. )

- **Review the report:**

    1. Do not hit ‘Submit’ as soon as you write the report. Review it at least once. Remove any typos.