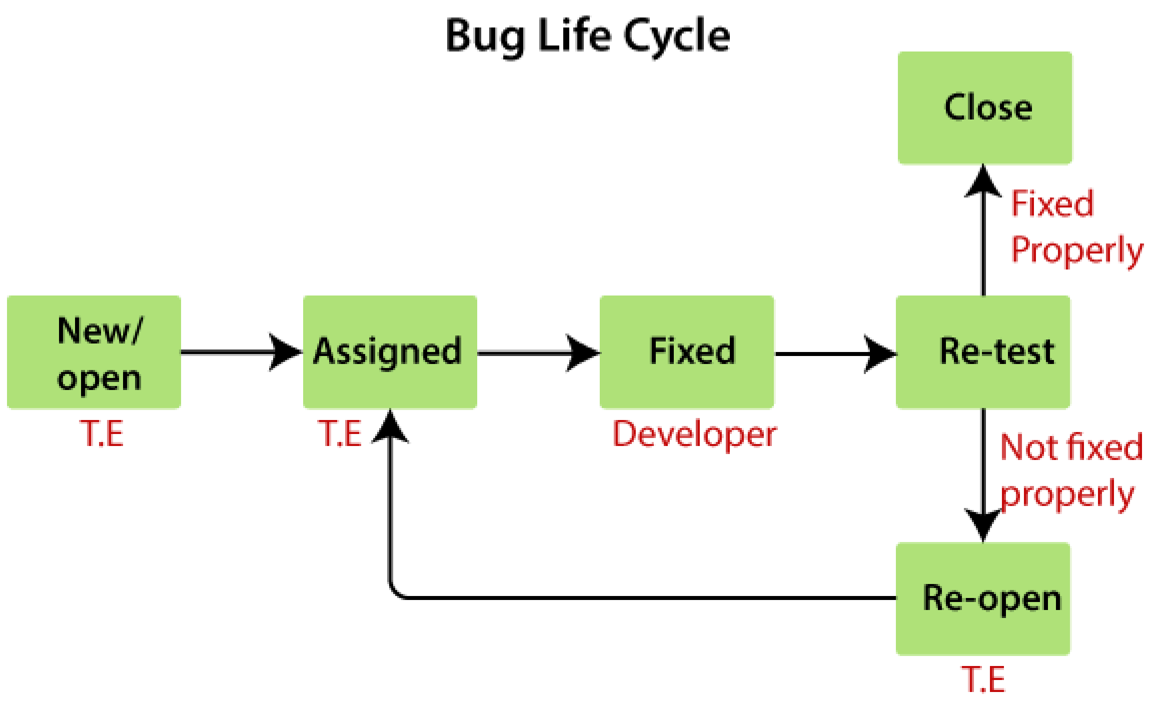
Bug Life Cycle



The bug life cycle, also known as the defect life cycle, describes the stages that a software bug goes through from the time it is identified until it is resolved. The bug life cycle typically consists of the following stages:

New:
This is the initial stage when the bug is first identified and reported by a tester or user.
The bug is in the "New" state until it is reviewed and confirmed by a developer or a quality assurance (QA) team member.
Open:
Once the bug is confirmed, it is assigned to a developer for further investigation and fixing.
The bug is in the "Open" state during the development phase.
Assigned:
The bug is assigned to a specific developer or a development team responsible for fixing it.
This stage indicates that the developer is actively working on the bug.

Fixed:
After the developer has successfully addressed the bug, the status is changed to "Fixed."
The bug is considered resolved in the developer's environment.
Retest:
In this stage, the QA team tests the fixed bug to ensure that it has been resolved successfully.
Closed:
The bug is marked as "Closed" when it has been fixed, retested, and verified.
This signifies that the bug has been resolved and the issue is considered closed.
Reopened:
If the bug is found to persist or reoccurs after being closed, it is reopened.
The bug goes back to the "Open" or "Assign" stage depending on the circumstances.
Duplicate:
If it's discovered that the reported issue is a duplicate of another bug, it may be marked as a duplicate.
The development team then focuses on fixing the original bug, and the duplicate is closed.
These stages may vary slightly depending on the specific bug tracking or issue management system used by a development team. The bug life cycle is crucial for effective communication between developers and testers and helps in tracking the status of bugs throughout the software development process.

Once assigned, developer can change status to:

- Duplicate defect
- Defect cannot be fixed
- It works fine for me/Issue not reproducible
- Postpone/Deferred

Duplicate defect

1) Same defect is logged by other Tester
2) Defect is already fixed

Defect cannot be fixed

1) When cost of defect is more than the actual defect
2) When technology itself is not supporting
3) When there is a minor defect in the root of the product

It works fine for me
1) Build mismatch
2) Platform mismatch
3) Improper defect report

Postpone/Deferred

1) When it is a minor defect and received during the end of the release.
2) When it is a defect and it is not affecting customer business workflow.
3) When it is a defect and customer does not want the feature in this release.
4) When it is a defect but customer is planning to do a lot of requirement changes in the particular feature.

Traceability matrix

It is also called as Requirement Traceability matrix(RTM) or Cross reference matrix (CRM)

It is a document through which we are ensuring that each and every requirement has minimum test cases.

Advantages of RTM:

1) Ensures complete requirements are documented.

2) It helps in analysing the root cause of any defect.

3) Applications can be developed according to the requirements.

Types of RTM:

1) Forward Traceability Matrix

2) Backward Traceability Matrix

3) Bi-directional Traceability Matrix

1) **Forward Traceability Matrix** – It is used to map requirements to the test cases. This is done before Test case execution. Here we are ensuring that the product developments are going in the right direction.

2) **Backward Traceability Matrix** – It is used to map Test cases to the requirement. This is done after Test case execution. Here we are ensuring that we are not going against/developing products against customer requirements.

3) **Bi-directional Traceability Matrix** – It is a combination of both Forward Traceability Matrix and Backward Traceability Matrix.

Test case Design Techniques

Types of Test case Design Techniques:

1) Error Guessing

2) Equivalence Class Partitioning

3) Boundary Value Analysis

4) Decision Table Technique

5) State Transition Diagram

1) **Error Guessing** – Here, TE will guess the error and derive more scenarios.

Example: Assume there is an “Amount” text field and the requirement says that it accepts + integer only.

Now we will have to enter only invalid inputs such as -10, abc, 10@ etc and try to guess more errors in this case.

2) **Equivalence Class Partitioning** - Here, when the input is in range of values, let us say, there is an “Amount” text field and the requirement says that it can accept numbers between 100 to 500. In this case what we can do is instead of entering all numbers, we can divide this range into equal classes I,e,. -100 to 0, 0 to 100, 100 to 200, 200 to 300, 300 to 400, 400 to 500 & 500 to 600. After this try entering any 1 value from each of this

class. Example if you enter 150 for the class 100 to 200, need not enter other values in range of 100 to 200, if it is accepting, then test case is pass, if it is not accepting, then test case is fail. Likewise we can only test only 2 scenarios/values(which includes 5 positive & 2 negative scenarios) and ensure that our test case coverage is achieved.

3) **Boundary Value Analysis** – Lets say, here you need values between the range A to B. We need tests for A,A+ & A- similarly B, B+ & B- . So here we will have 4 positive scenarios and 2 negative scenarios. Since the boundaries are covered, our test case coverage is good and no need to test for other values.

4) **Decision Table Technique** – In this Technique, we check for multiple conditions, combinations and Rule criterias.

Formula = 2^no of conditions= total no of rules or scenarios

Example1:

Requirement- Customer wants to order from Swiggy,

1) First time customers get 50% discount
2) If the coupon code is used, they get a 25% discount.

In this case there are 2 conditions, 2^2 = 4 scenarios we can derive.

Example 2 for Login page: (Assignment)

5) **State Transition Diagram** – This technique is used to check for different screens of a software. It is basically a pictorial representation of the scenarios.

Example: Let's say a person has to withdraw cash from an ATM machine, we can derive 4 scenarios for this.

1) When he enters the correct pin for the first time, he withdraws the cash.
2) When he enters the wrong pin for the first time and the correct pin for the second time, he can withdraw the cash.

3) When he enters the wrong pin for the first and second time and enters the correct pin for the third time, he can withdraw the cash.

4) When he enters the wrong pin for all attempts i,e,. first, second & third attempt, the card gets blocked and he cannot withdraw the cash.

These scenarios customers can write in a pictorial way and present how scenarios we can derive here.