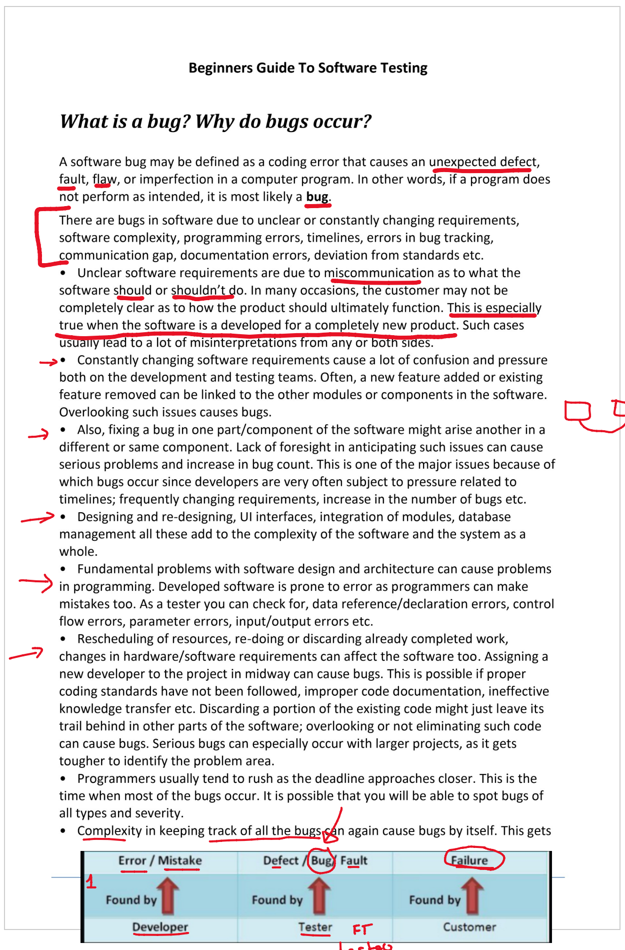
SESSION 3



What is a bug? Why do bugs occur?

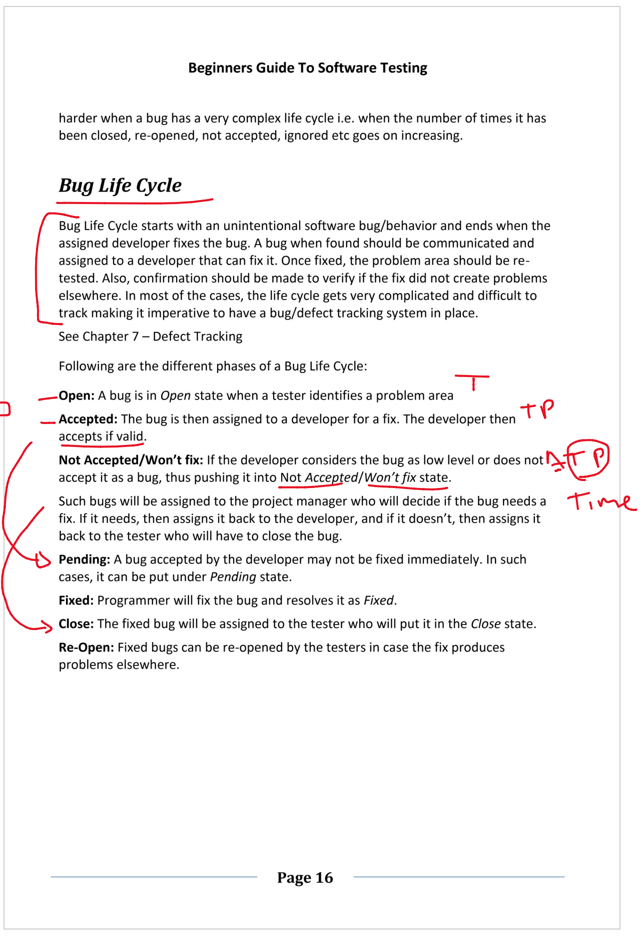
A software bug may be defined as a coding error that causes an unexpected defect, fault, flaw, or imperfection in a computer program. In other words, if a program does not perform as intended, it is most likely a **bug**.

There are bugs in software due to unclear or constantly changing requirements, software complexity, programming errors, timelines, errors in bug tracking, communication gap, documentation errors, deviation from standards etc.

- Unclear software requirements are due to miscommunication as to what the software should or shouldn't do. In many occasions, the customer may not be completely clear as to how the product should ultimately function. This is especially true when the software is a developed for a completely new product. Such cases usually lead to a lot of misinterpretations from any or both sides.
- Constantly changing software requirements cause a lot of confusion and pressure both on the development and testing teams. Often, a new feature added or existing feature removed can be linked to the other modules or components in the software. Overlooking such issues causes bugs.
- Also, fixing a bug in one part/component of the software might arise another in a different or same component. Lack of foresight in anticipating such issues can cause serious problems and increase in bug count. This is one of the major issues because of which bugs occur since developers are very often subject to pressure related to timelines; frequently changing requirements, increase in the number of bugs etc.
- Designing and re-designing, UI interfaces, integration of modules, database management all these add to the complexity of the software and the system as a whole.
- Fundamental problems with software design and architecture can cause problems in programming. Developed software is prone to error as programmers can make mistakes too. As a tester you can check for, data reference/declaration errors, control flow errors, parameter errors, input/output errors etc.
- Rescheduling of resources, re-doing or discarding already completed work, changes in hardware/software requirements can affect the software too. Assigning a new developer to the project in midway can cause bugs. This is possible if proper coding standards have not been followed, improper code documentation, ineffective knowledge transfer etc. Discarding a portion of the existing code might just leave its trail behind in other parts of the software; overlooking or not eliminating such code can cause bugs. Serious bugs can especially occur with larger projects, as it gets tougher to identify the problem area.
- Programmers usually tend to rush as the deadline approaches closer. This is the time when most of the bugs occur. It is possible that you will be able to spot bugs of all types and severity.
- Complexity in keeping track of all the bugs can again cause bugs by itself. This gets

Error / Mistake	Defect / Bug / Fault	Failure
1 Found by	Found by	Found by
Developer	Tester FT	Customer

testas Formales
Doc
Futures
Seg



harder when a bug has a very complex life cycle i.e. when the number of times it has been closed, re-opened, not accepted, ignored etc goes on increasing.

Bug Life Cycle

Bug Life Cycle starts with an unintentional software bug/behavior and ends when the assigned developer fixes the bug. A bug when found should be communicated and assigned to a developer that can fix it. Once fixed, the problem area should be re-tested. Also, confirmation should be made to verify if the fix did not create problems elsewhere. In most of the cases, the life cycle gets very complicated and difficult to track making it imperative to have a bug/defect tracking system in place.

See Chapter 7 – Defect Tracking

Following are the different phases of a Bug Life Cycle:

Open: A bug is in *Open* state when a tester identifies a problem area T

Accepted: The bug is then assigned to a developer for a fix. The developer then accepts if valid. TP

Not Accepted/Won't fix: If the developer considers the bug as low level or does not accept it as a bug, thus pushing it into Not *Accepted*/*Won't fix* state. ATP

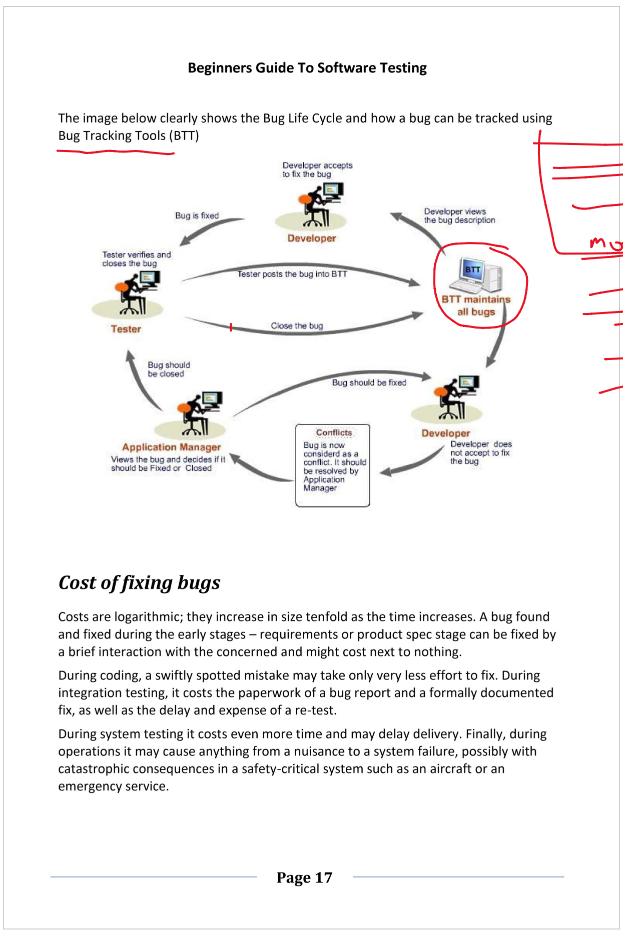
Such bugs will be assigned to the project manager who will decide if the bug needs a fix. If it needs, then assigns it back to the developer, and if it doesn't, then assigns it back to the tester who will have to close the bug. Time

Pending: A bug accepted by the developer may not be fixed immediately. In such cases, it can be put under *Pending* state.

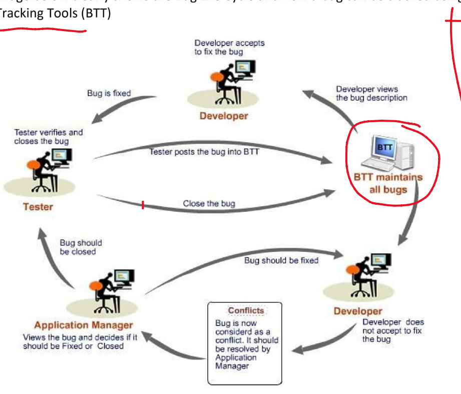
Fixed: Programmer will fix the bug and resolves it as *Fixed*.

Close: The fixed bug will be assigned to the tester who will put it in the *Close* state.

Re-Open: Fixed bugs can be re-opened by the testers in case the fix produces problems elsewhere.



The image below clearly shows the Bug Life Cycle and how a bug can be tracked using Bug Tracking Tools (BTT)


Developer accepts to fix the bug
Developer
Bug is fixed
Developer views the bug description
Tester verifies and closes the bug
Tester posts the bug into BTT
BTT
BTT maintains all bugs
Tester
Close the bug
Bug should be closed
Bug should be fixed
Developer
Developer does not accept to fix the bug
Conflicts
Bug is now considerd as a conflict. It should be resolved by Application Manager
Application Manager
Views the bug and decides if it should be Fixed or Closed

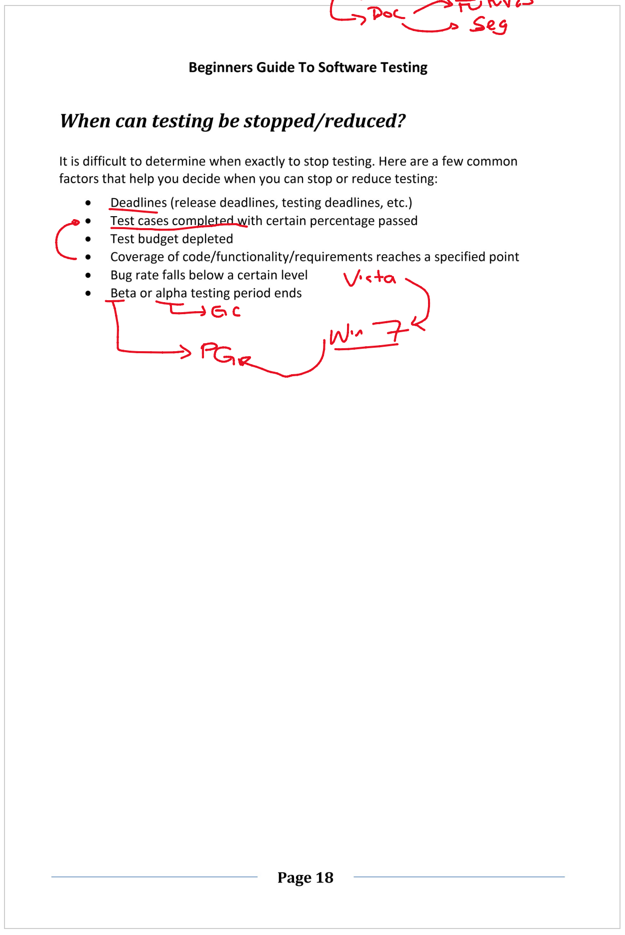
more

Cost of fixing bugs

Costs are logarithmic; they increase in size tenfold as the time increases. A bug found and fixed during the early stages – requirements or product spec stage can be fixed by a brief interaction with the concerned and might cost next to nothing.

During coding, a swiftly spotted mistake may take only very less effort to fix. During integration testing, it costs the paperwork of a bug report and a formally documented fix, as well as the delay and expense of a re-test.

During system testing it costs even more time and may delay delivery. Finally, during operations it may cause anything from a nuisance to a system failure, possibly with catastrophic consequences in a safety-critical system such as an aircraft or an emergency service.

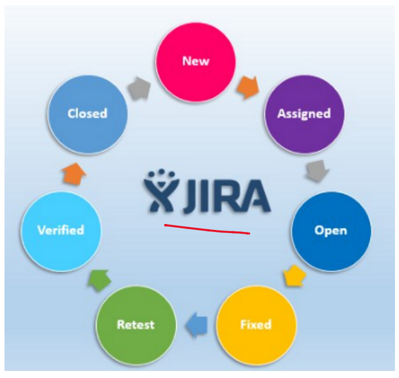


When can testing be stopped/reduced?

It is difficult to determine when exactly to stop testing. Here are a few common factors that help you decide when you can stop or reduce testing:

- Deadlines (release deadlines, testing deadlines, etc.)
- Test cases completed with certain percentage passed
- Test budget depleted
- Coverage of code/functionality/requirements reaches a specified point
- Bug rate falls below a certain level
- Beta or alpha testing period ends

GC
PGR
Vista
Win 7



HW3 - DEFINITIONS:

- ERROR
- BUG
- FAILURE

- VERIFICATION
- VALIDATION

Tuesday 11pm - Classroom