INTRODUCTION

SOFTWARE TESTING

Video Link on Lecture Discussion:

https://web.microsoftstream.com/video/db79e775-a1c0-4b6d-be9a-14ae43d4fa94

What is testing?

- Testing is the process of running a program against "test cases" in order to get some evidence of the correctness of the program.
- If an error is discovered, then the process of debugging attempts to locate the error and fix it.

Definitions of “TESTING”

- Hetzel: Any activity aimed at evaluating an attribute or capability of a program or system. It is the measurement of software quality.
- Beizer: The act of executing tests. Tests are designed and then executed to demonstrate the correspondence between an element and its specification.

Definitions of “TESTING” (cont’d)

- Myers: The process of executing a program with the *intent* of finding errors.
- IEEE: The process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements or to identify differences between expected and actual results.

Definitions of “TESTING” (cont’d)

- Testing undertaken to *demonstrate that a system performs correctly* is sometimes referred to as **validation testing**.
- Testing undertaken to *expose defects* is sometimes referred to as **defect testing**.

Difficulty of Testing

- Consider a 100 line program:
  - Contains 2 nested loops that execute 20 times each
  - Inside the interior loop there are 4 if-then-else statements
- There are approx. 10^{14} possible paths that may be executed in this program
- On a representative processor, testing all paths would require 3170 years (24x7x365)

Limitations of Testing

- Can show the presence of errors
- Cannot confirm the absence of errors
- Research going on to develop “provably correct” code

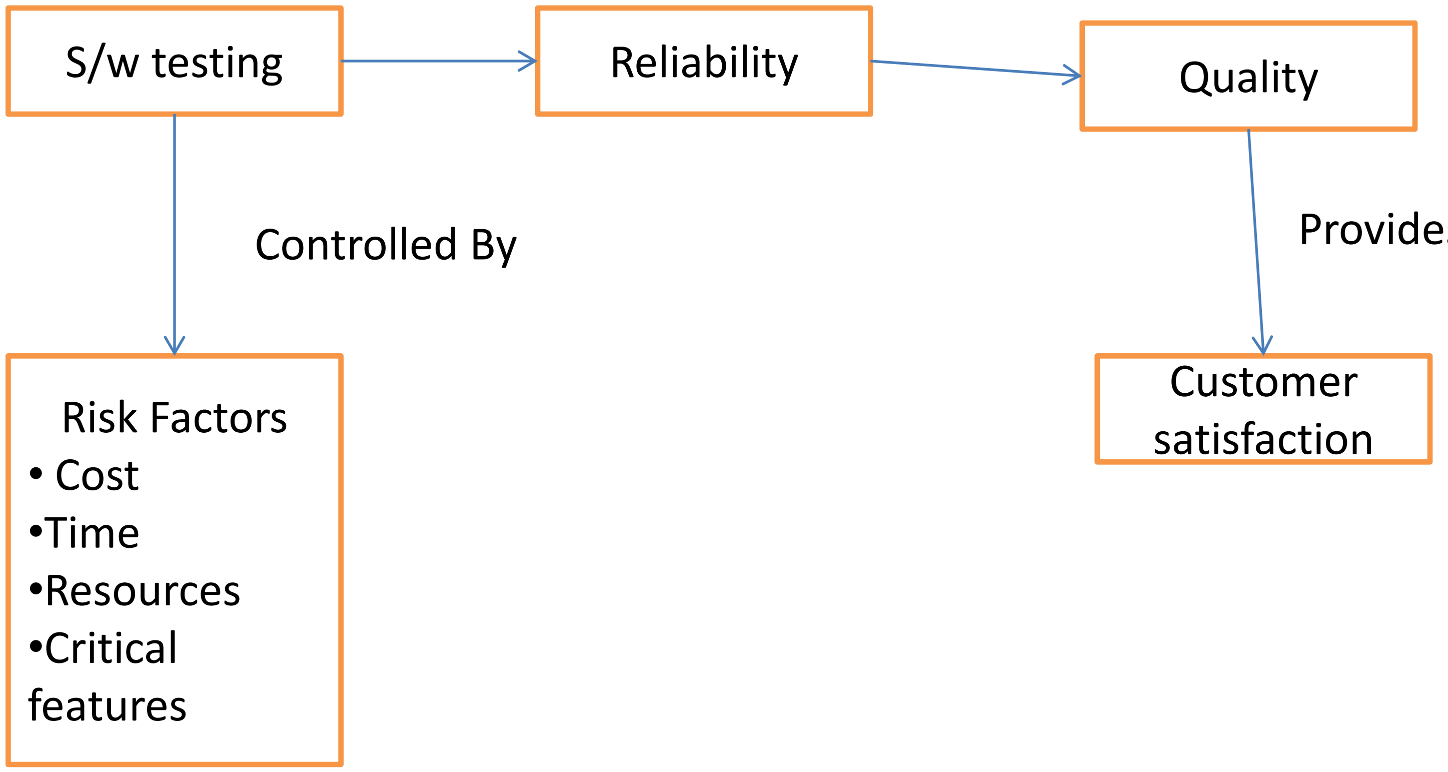
Software Testing-Myths and Facts

- Myth-Testing is a single phase in SDLC.
- Myth- Testing is easy.
- Myth- Software development is worth more than testing.
- Myth- Complete testing is possible.
- Myth-Testing starts after program development.
- Myth- The purpose of testing is to check the functionality of the software.
- Myth-Anyone can be a tester.

Goals of Software Testing

- Short term or Immediate Goals
  - Bug discovery
  - Bug prevention
- Long Term Goals
  - Reliability i.e. s/w will not fail . i.e. rigorous testing
  - Quality
  - Customer satisfaction i.e. cover specified and unspecified requirements covered.
  - Risk management i.e. undesirable events (time, cost, resource, critical features)
- Post implementation Goals
  - Reduced maintenance cost
  - Improved testing process
-

Long term goals snapshot



Psychology for Software Testing

- Testing is the process of demonstrating that there are no errors.
- Testing is the process of executing a program with the intent of finding errors.

Let’s Pause for a Moment…

Imagine that it’s summertime and that a 3-day weekend is just starting… Wouldn’t it be great to just grab a fishin’ pole and head on out to the lake!

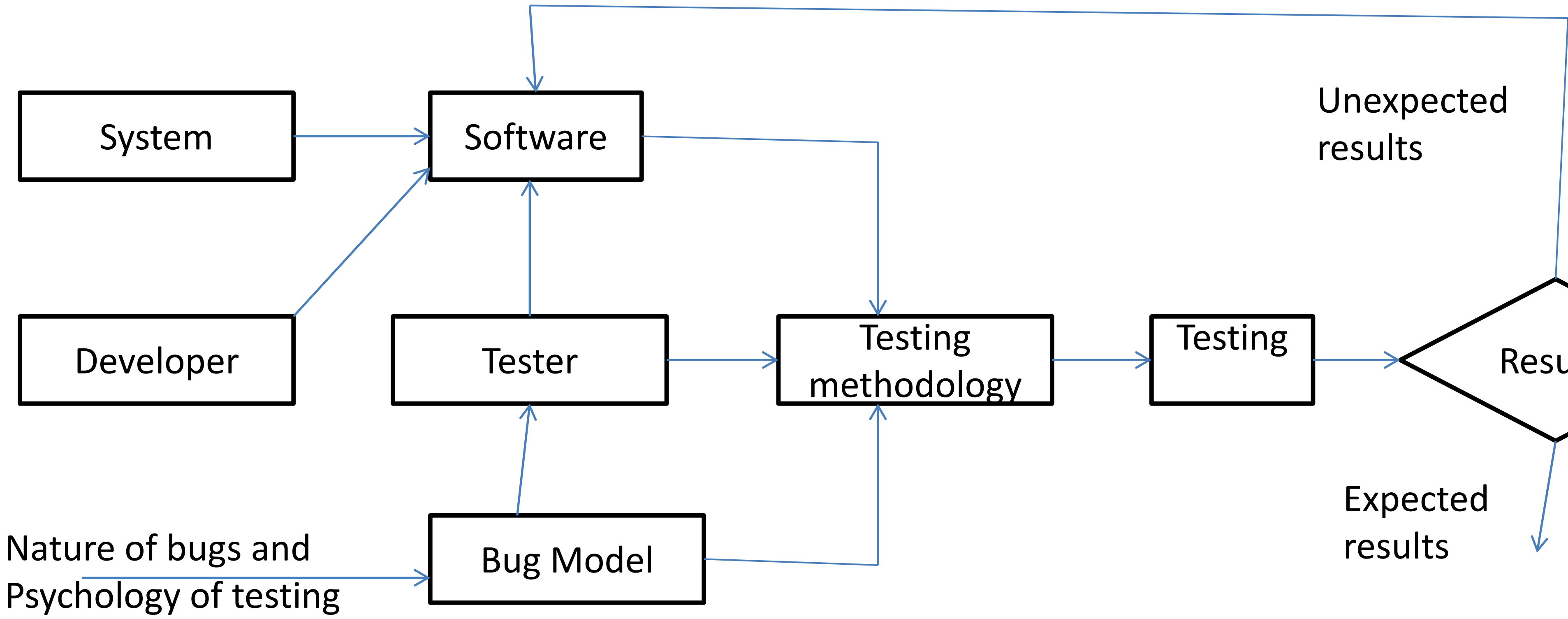
Fisherman’s Dilemma

- You have 3 days for fishing and 2 lakes to choose from. Day 1 at lake X nets 8 fish. Day 2 at lake Y nets 32 fish. Which lake do you return to for day 3?
- Does your answer depend on any assumptions?

Di Lemma

- In general, the probability of the existence of more errors in a section of a program is directly related to the number of errors already found in that section.

Model for software testing

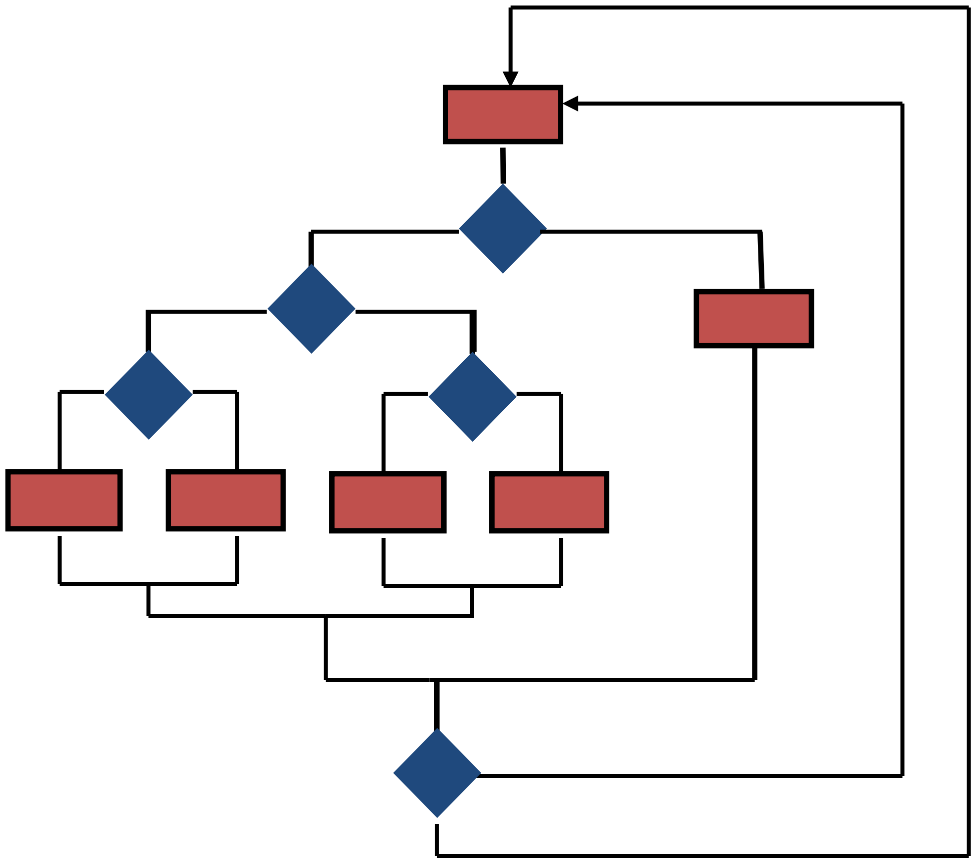


Effective (selective) S/W Testing Vs. Exhaustive (Complete) Software Testing

- Exhausting testing means that every statement in the program and every possible path combination with every possible combination of data must be executed.
- But, it is out of scope:
- Number of questions arise:
- When are we done with testing?
- How do we know that we have tested enough?
- Etc.

Solution is effective testing

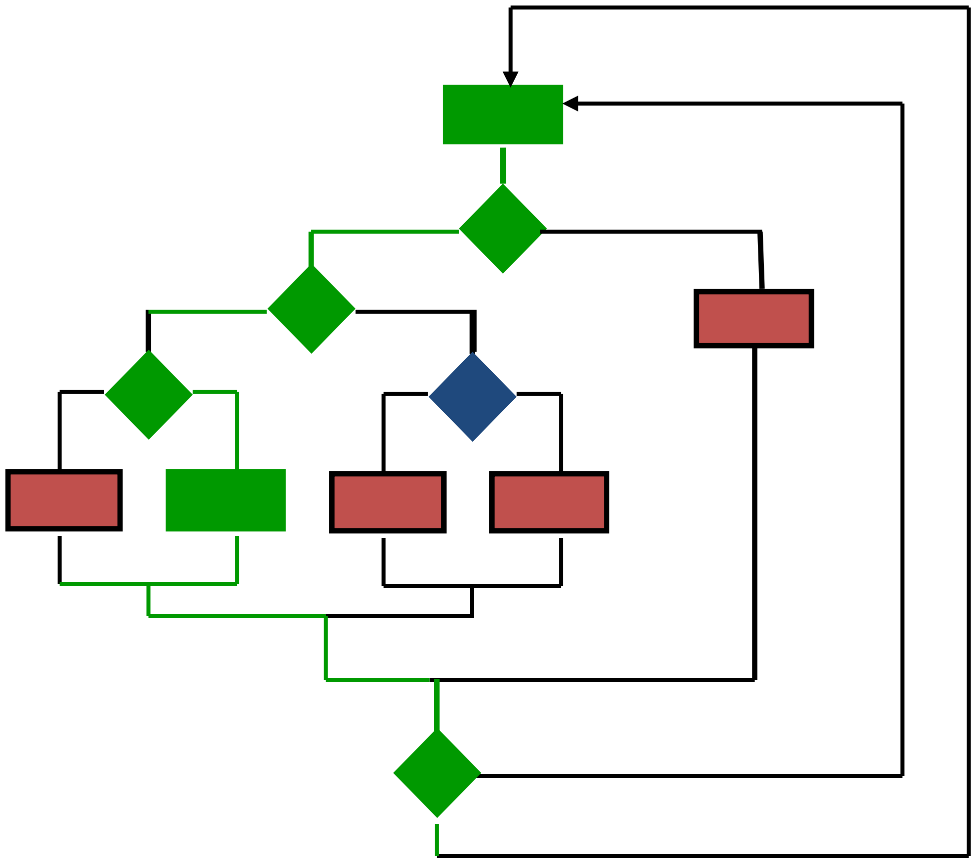
Exhaustive Testing (infeasible)



Two nested loops containing four if..then..else statements. Each loop can execute up to 20 times

There are 10^14 possible paths! If we execute one test per millisecond, it would take 3170 years to test this program

Selective Testing (feasible)

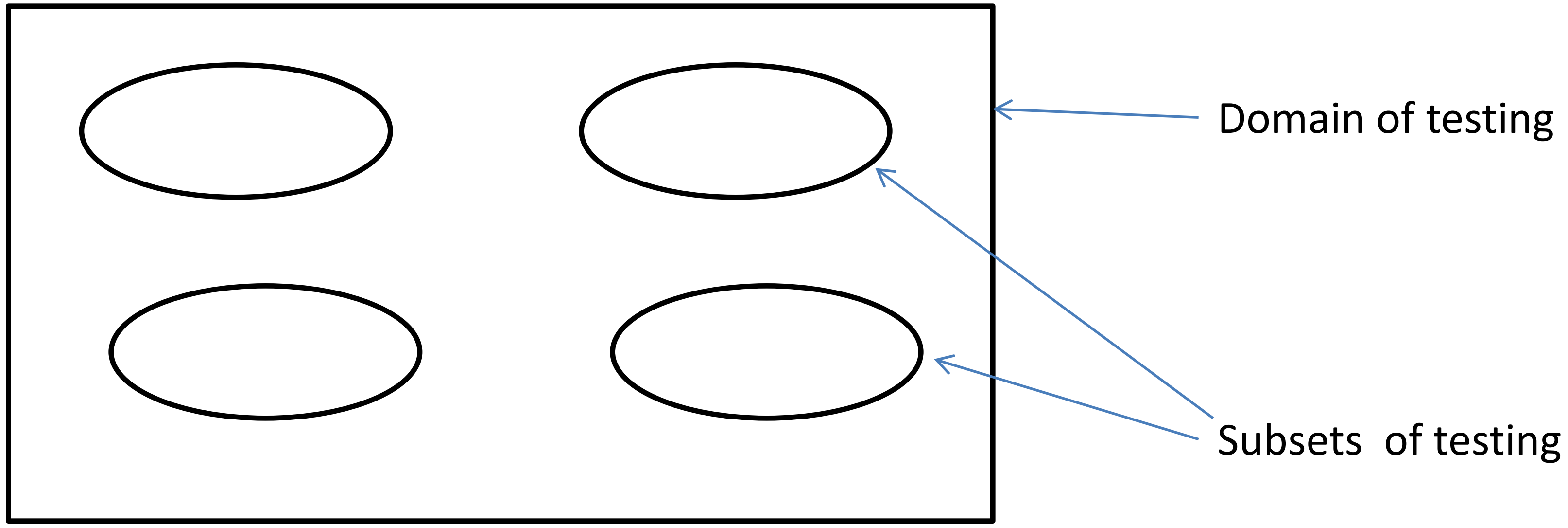


Test a carefully selected execution path. Cannot be comprehensive

Why complete testing is not possible

1. The Domain of possible inputs to the software is too large to test
   - Valid inputs
   - Invalid inputs
   - Edited inputs
   - Race condition inputs
2. There are many possible through the program to test
3. Every design error cannot be found

1. The Domain of possible inputs to the software is too large to test



1. The Domain of possible inputs to the software is too large to test

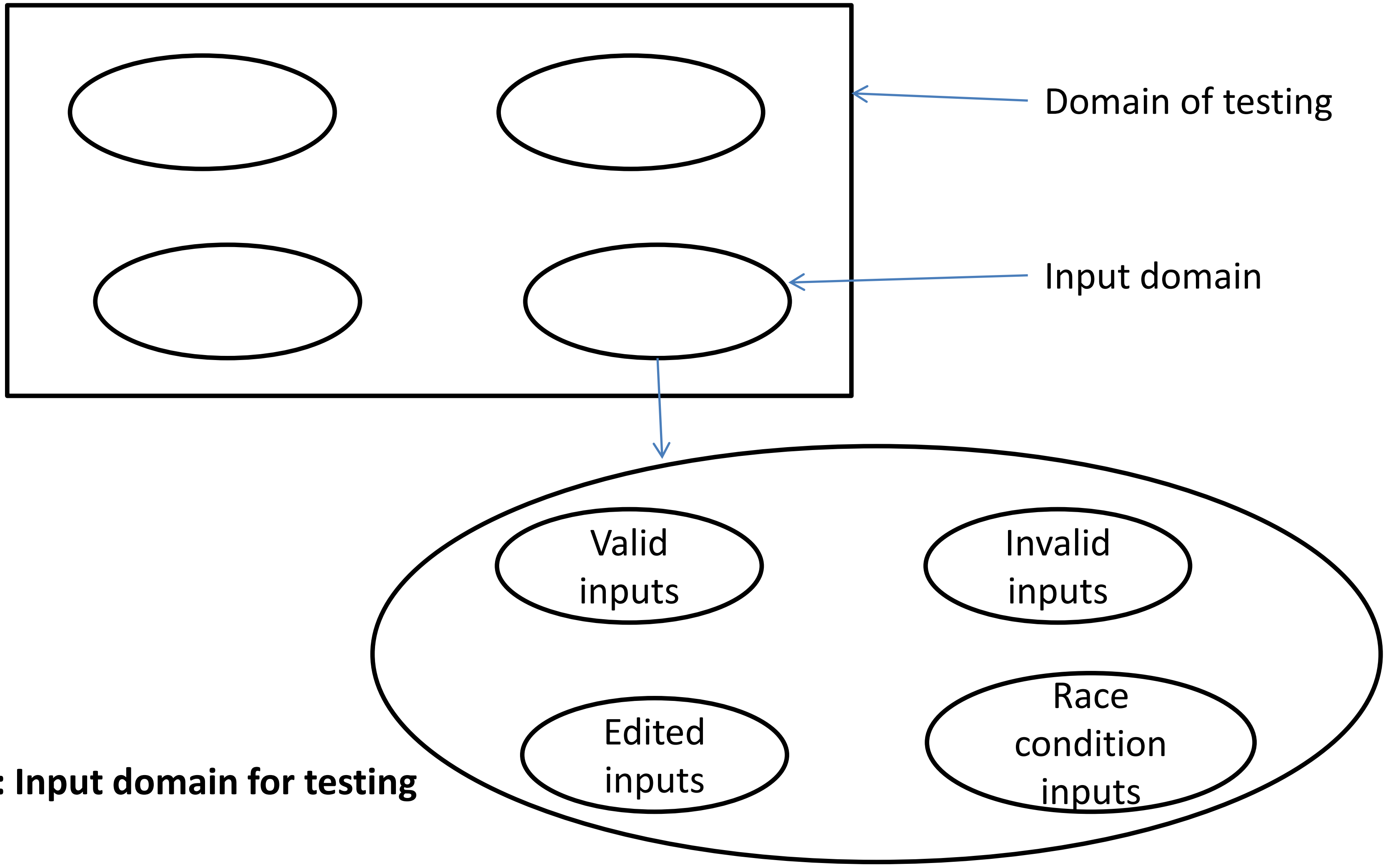


Fig: Input domain for testing

Valid inputs

- Consider a simple addition of two digit two numbers.
- Range is -99 to 99 (total 199)
- Total no of possible combinations= 199x199=39601

Invalid Inputs

- To observe the behavior of the program as to how it responds when a user feeds invalid inputs.
- It is too large to test.
- In last example,
- Number out of range
- Combination of alphabets and digits
- Combination of all alphabets
- Combination of control characters
- Combination of any key other key on keyboard

Edited inputs

- Edit input at the time of providing inputs to the program, then many unexpected input events may occur.
- E.g.
- 1. add many spaces in the input, which are not visible to the user.
  - It can be a reason for non-functioning of the program.
- 2. a user is pressing a number key, then backspace key continously and finally after sometime, he presses another number key and enter.
  - Its input buffer overflows and the system crashes.
- The behavior of the system can not be judged. They can behave in a number of ways, causing defect in testing a program.
- Edited inputs can test completely.

Race conditions

- The timing variation between two or more inputs is also one of the issues that limit the testing.
- E.g.
  - There are two input events, A and B.
  - According to the design, A precede B in most of the cases. But, B can also come first in rare and restricted conditions.
  - This is race condition, whenever B precedes A.
- Generate race condition bug.
- In case of multiprocessing systems, interactive systems.
- Race conditions are among least tested.

2. There are many possible through the program to test

- A program path can be traced through the code from the star of a program to its termination.
- Two paths differ if the program executes different statements in each, or executes the same statements but in different order.
- A testing person thinks that if all the possible paths of control flow through the program are executed, then possibly the program can be said to be completely tested.
- Explain here difficulty of testing.

Testing Vs Debugging

- Testing is focused on identifying the problems in the product
- Done by Tester
- Need not know the source code

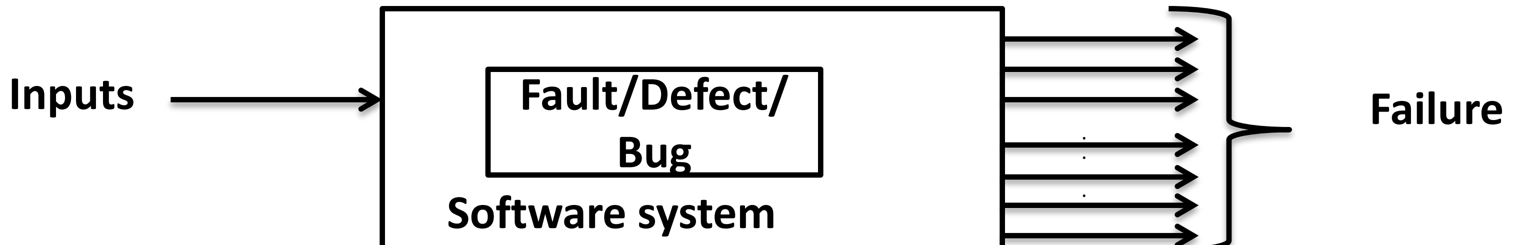
- Debugging is to make sure that the bugs are removed or fixed
- Done by Developer
- Need to know the source Code

Terminology

- Failure
- Fault/defect/ bug
- Error
- Relationships among Failure, fault, and error
- Test case
- Testware, Incident & Oracle

Failure, Fault/Defect/Bug

- Failure:
  - When results or behavior of the system under test are different as compared to specified expectations.
  - The inability of a system or component to perform a required function according to its specification.
- Fault/Defect/ Bug: synonyms (Problems related to O/P side.)
  - A condition that in actual causes a system to produce failure.
  - Bug can be defined as the **abnormal behavior of** the software.
  - **No software exists without a bug.**
  - The **elimination of bugs** from the software depends upon **the efficiency of testing** done on the software.
  - A bug is a specific concern about **the quality of the Application under Test** (AUT).
- Fault & Failure
  - Fault is the reason embedded in any phase of SDLC and results in failures.
  - When bug is executed, then failures are generated. (Not true always)
  - Some bugs are hidden i.e. not executed sometime due to not get the required condition.
  - i.e. hidden bugs not always produce failures.

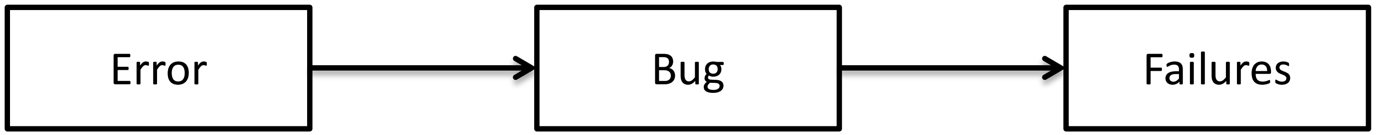


Error

- Very general term used for human mistakes.
- Mistake in any phase of SDLC, produced errors.
- Types
  - Typographical error
  - Misleading of a specification
  - Misunderstanding of what a subroutines does etc.

Error, Bug, Failures

- An error causes a bug and then bug in turn causes failures.



Flow of faults

Exercise

- Consider the following module in a software:

```
Module A( )
{
……………………………..
        while( index <= number - 1 );
        {
        if( number % index == 0 )
        {
                                    printf( "Not a prime number" );
                                    break;
        }
        index++;
    }
    if( index == number )
       printf( "Prime number" );
    getch( );
}
```

1. Identify when it will produce failure of the program?
2. Reason of the failure?
3. What is bug/defect/fault here?

Test Case

- Objective: To find errors in the system.
- A well documented procedure designed to **test the functionality of a feature** in the system.
- It has an identity and associated with a program behavior.
- It accepts set of inputs and its corresponding expected outputs.

Test Case ID
Purpose
Preconditions
Inputs
Expected Outputs

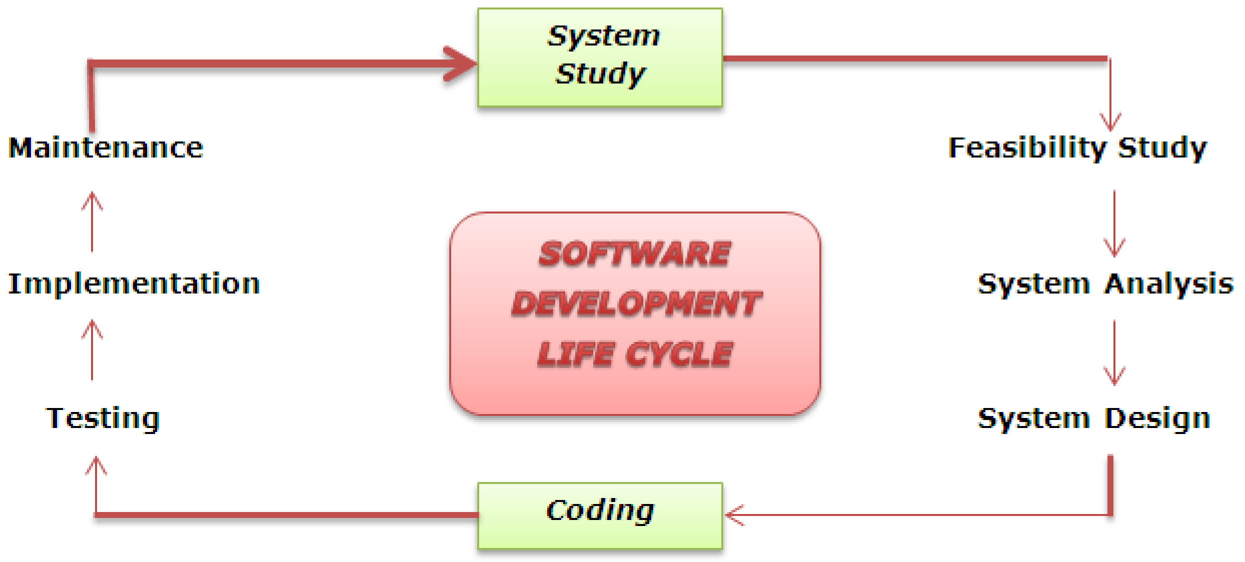
Test case template

Test case

- A good test case is one that has a probability of finding an as yet undiscovered error.
- A successful test is one that uncovers a yet undiscovered error.
- A good test is not redundant.
- A good test should be “best of breed”.
- A good test should neither be too simple nor too complex.
- Create test cases in such a way that testing is done to uncover the hidden bugs and also ensure that the software is usable and reliable

Testware, Incident & Oracle

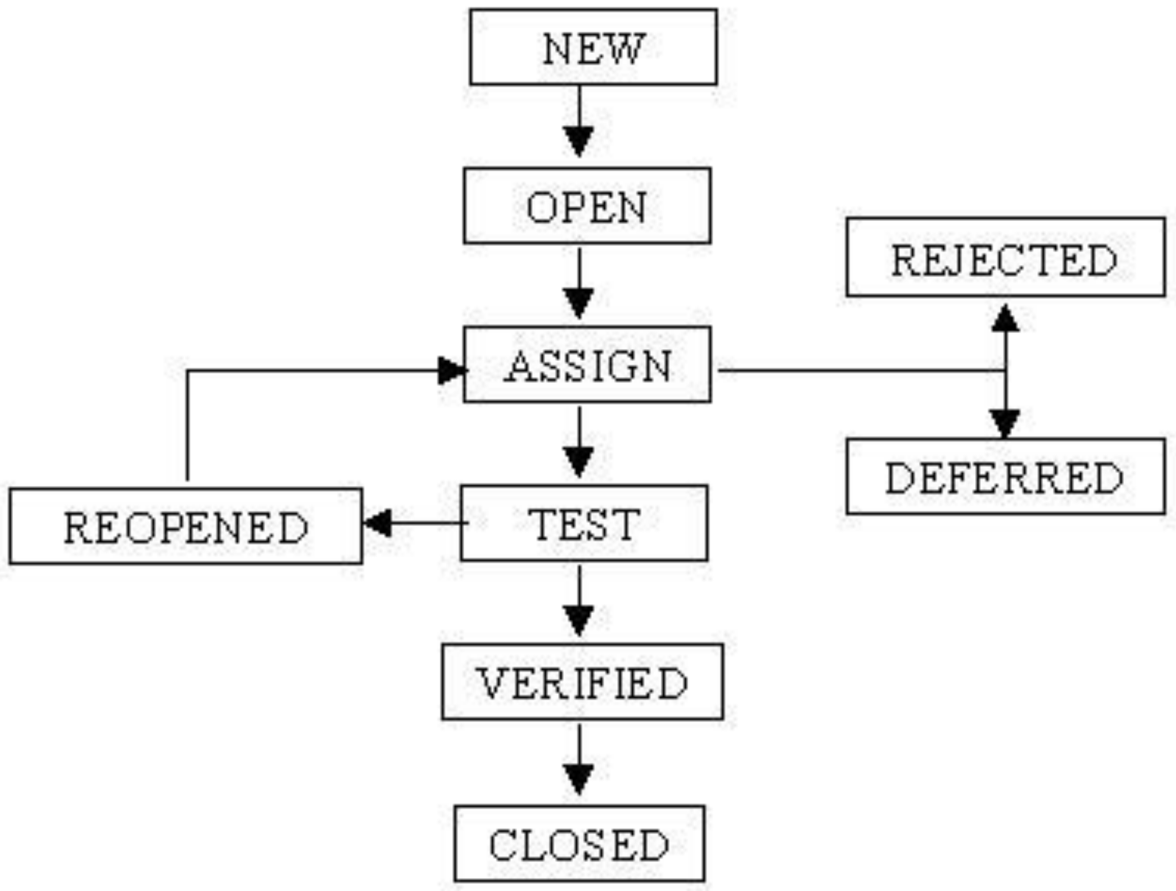
- Testware
  - Analogy from software and hardware
  - Testware are the documents that a test engineer produces.
  - Testware documents should also be managed and updated like a s/w product.
  - It contain
    - Test plans
    - Test specifications
    - Test case design
    - Test reports
- **Incident**
  - The symptom(s) associated with a failure that alerts the user about the occurrence of a failure.
- Test Oracle
  - Means **to judge the success or failure of a test**, i.e. **To judge the correctness of the system for some test**.
  - **The simplest oracle** is the comparing actual results by hand.
  - **The automated oracle** is software tool based. It reduce the overhead of simplest oracle.



PHASES OF SYSTEM DEVELOPMENT LIFE

Life cycle of a Bug

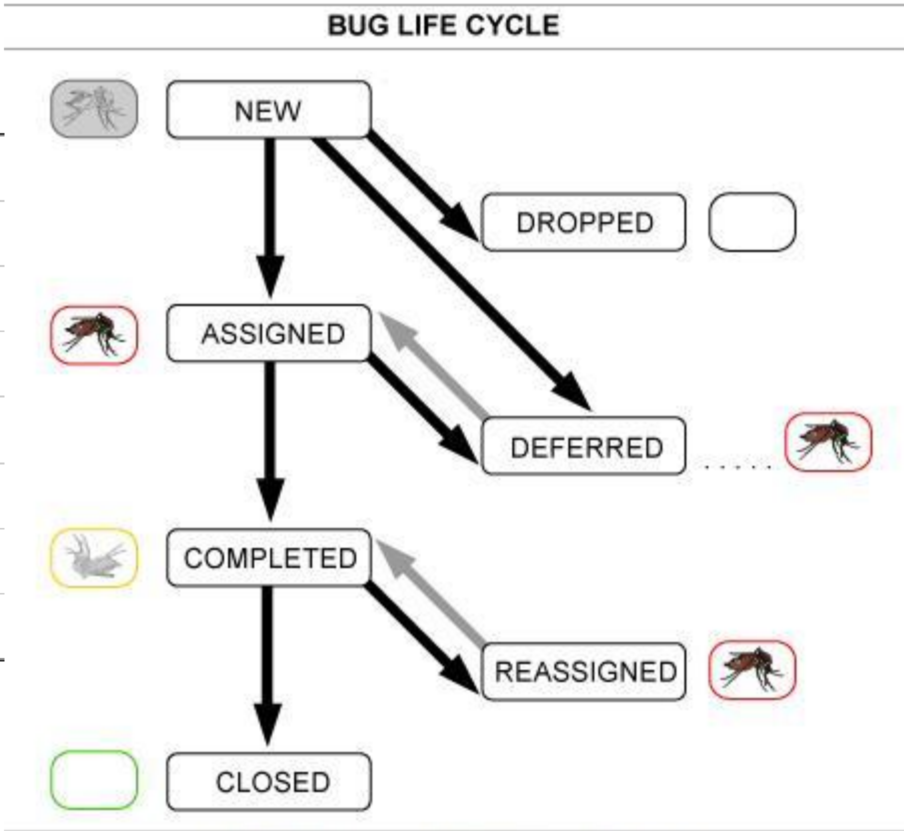
- Software Bug / Defect Life Cycle is the journey of a bug from its identification to its closure.
- In software development process, the bug has a life cycle. The bug should go through the life cycle to be closed.
- A specific life cycle ensures that the process is standardized.
- The bug attains different states in the life cycle.
- The Life Cycle varies from organization to organization and is governed by the software testing process the organization / project follows and/or the Bug / Defect tracking tool being used
- The different states of a bug can be summarized as follows:
- 1. New
  2. Open
  3. Assign
  4. Test
  5. Verified
  6. Deferred
  7. Reopened
  8. Duplicate
  9. Rejected and
  10. Closed

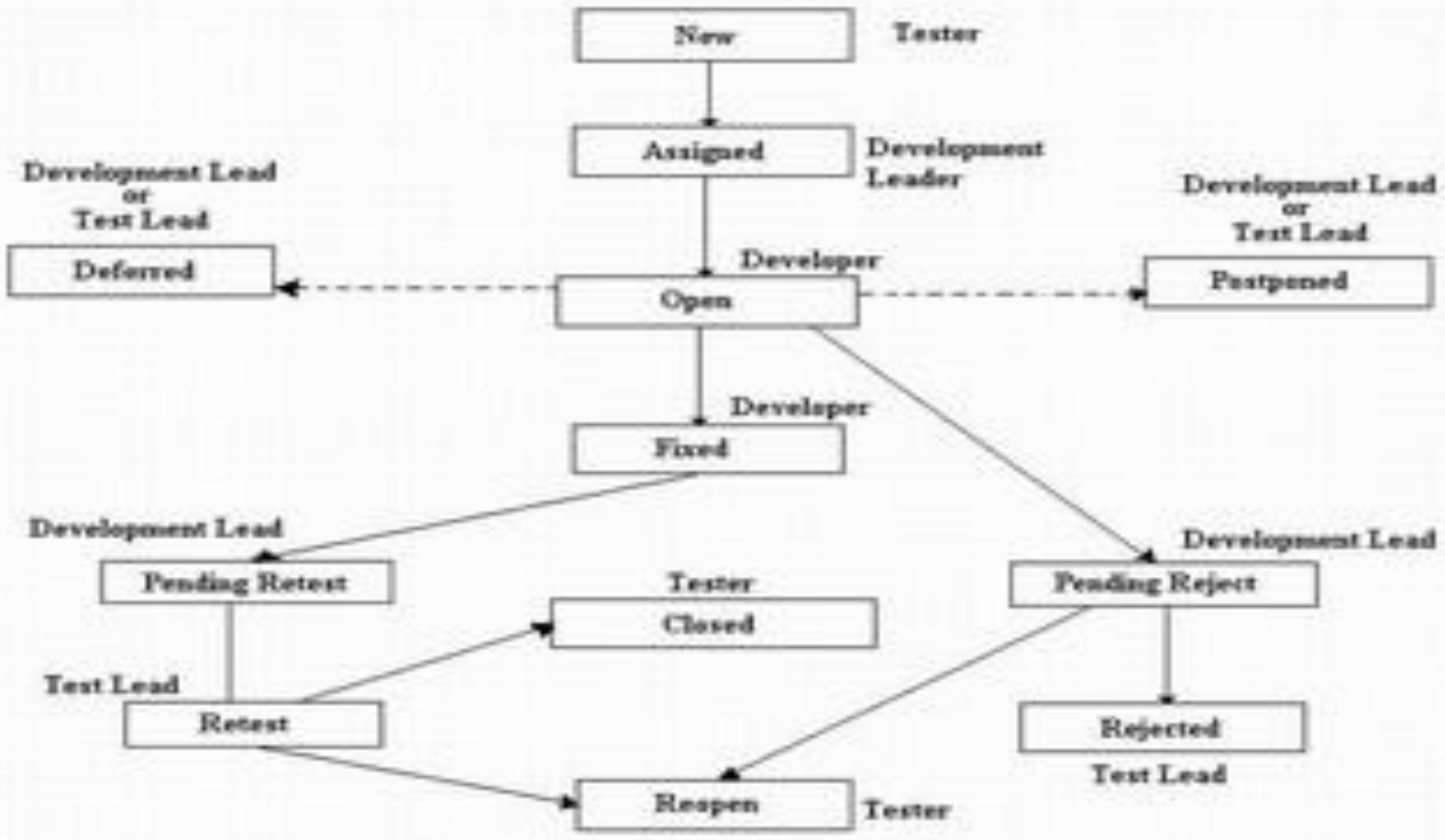

NEW
OPEN
ASSIGN
REJECTED
DEFERRED
TEST
REOPENED
VERIFIED
CLOSED

1. **New:** When the bug is reported for the first time, its state will be “NEW”.
2. **Open:** After a QA has reported a bug, the lead of the tester approves that the bug is genuine and he changes the state as “OPEN”.
3. **Assign:** Once the lead changes the state as “OPEN”, he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to “ASSIGN”.
4. **Test:** Once the developer fixes the bug, he reassign the bug to the testing team for next round of verification. 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.
5. **Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. There can be ‘n’ number of reasons for changing the bug to this state. 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 etc.
6. **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”.
7. **Duplicate :** If the same bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to “DUPLICATE”.
8. **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”.
9. **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.
10. **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.

Bug Life Cycle

Status	Alternative Status
NEW	
ASSIGNED	OPEN
DEFERRED	
DROPPED	REJECTED
COMPLETED	FIXED, RESOLVED, TEST
REASSIGNED	REOPENED
CLOSED	VERIFIED





Bug Life Cycle

- *NEW*: Tester finds a ‘bug’ and posts it with the status NEW. This bug is yet to be studied/approved. The fate of a NEW bug is one of ASSIGNED, DROPPED and DEFERRED.
- *ASSIGNED / OPEN*: Test / Development / Project lead studies the NEW bug and if it is found to be valid it is assigned to a member of the Development Team. The assigned Developer’s responsibility is now to fix the bug and have it COMPLETED. Sometimes, ASSIGNED and OPEN can be different statuses. In that case, a bug can be open yet unassigned.
- *DEFERRED*: If a valid NEW or ASSIGNED bug is decided to be fixed in upcoming releases instead of the current release it is DEFERRED. This bug is ASSIGNED when the time comes.
- *DROPPED / REJECTED*: Test / Development/ Project lead studies the NEW bug and if it is found to be invalid, it is DROPPED / REJECTED. Note that the specific reason for this action needs to be given.

- *COMPLETED / FIXED / RESOLVED / TEST*: Developer ‘fixes’ the bug that is ASSIGNED to him or her. Now, the ‘fixed’ bug needs to be verified by the Test Team and the Development Team ‘assigns’ the bug back to the Test Team. A COMPLETED bug is either CLOSED, if fine, or REASSIGNED, if still not fine.
- If a Developer cannot fix a bug, some organizations may offer the following statuses:
  - *Won’t Fix / Can’t Fix*: The Developer will not or cannot fix the bug due to some reason.
  - *Can’t Reproduce*: The Developer is unable to reproduce the bug.
  - *Need More Information*: The Developer needs more information on the bug from the Tester.
- *REASSIGNED / REOPENED*: If the Tester finds that the ‘fixed’ bug is in fact not fixed or only partially fixed, it is reassigned to the Developer who ‘fixed’ it. A REASSIGNED bug needs to be COMPLETED again.
- *CLOSED / VERIFIED*: If the Tester / Test Lead finds that the bug is indeed fixed and is no more of any concern, it is CLOSED / VERIFIED. This is the happy ending.

S/W Testing Life Cycle: Phases

- Consists of a series of stages through which a software product goes through and describes the various activities pertaining to testing that are carried out on the product.
- To ensure that the software meets the specifications of the end user and is the software what the end user actually wanted.
- To identify, which are the testing activities to be carried out and when should they be carried out.
- The method in which the software testing activities are carried out will depend on how was the software developed.
- In some cases, the entire software is developed at one go while in some, it may be developed in small parts.
- The phases of testing life cycle remain more or less the same.
- The life cycle of testing process intersects the software development lifecycle.

Software Error: Categories

- User interface errors such as output errors or incorrect user messages.
- Function errors
- Hardware defects
- Incorrect program version
- Requirements errors
- Design errors
- Documentation errors
- Architecture errors
- Module interface errors
- Performance errors
- Boundary-related errors
- Logic errors such as calculation errors, State-based behavior errors, Communication errors, Program structure errors, such as control-flow errors.

Classification of Bugs

- Various ways in which we can classify
  - Status wise: states of bugs: Open, Close, Deferred, Cancelled etc.
  - Severity Wise: Major, Minor, Fatal
  - Work product wise: SSD, FSD, ADS, DDS, Source Code, Test Plan/ Test Cases, User Documentation
  - Errors Wise: 24 or above

Classification of Bugs

- **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

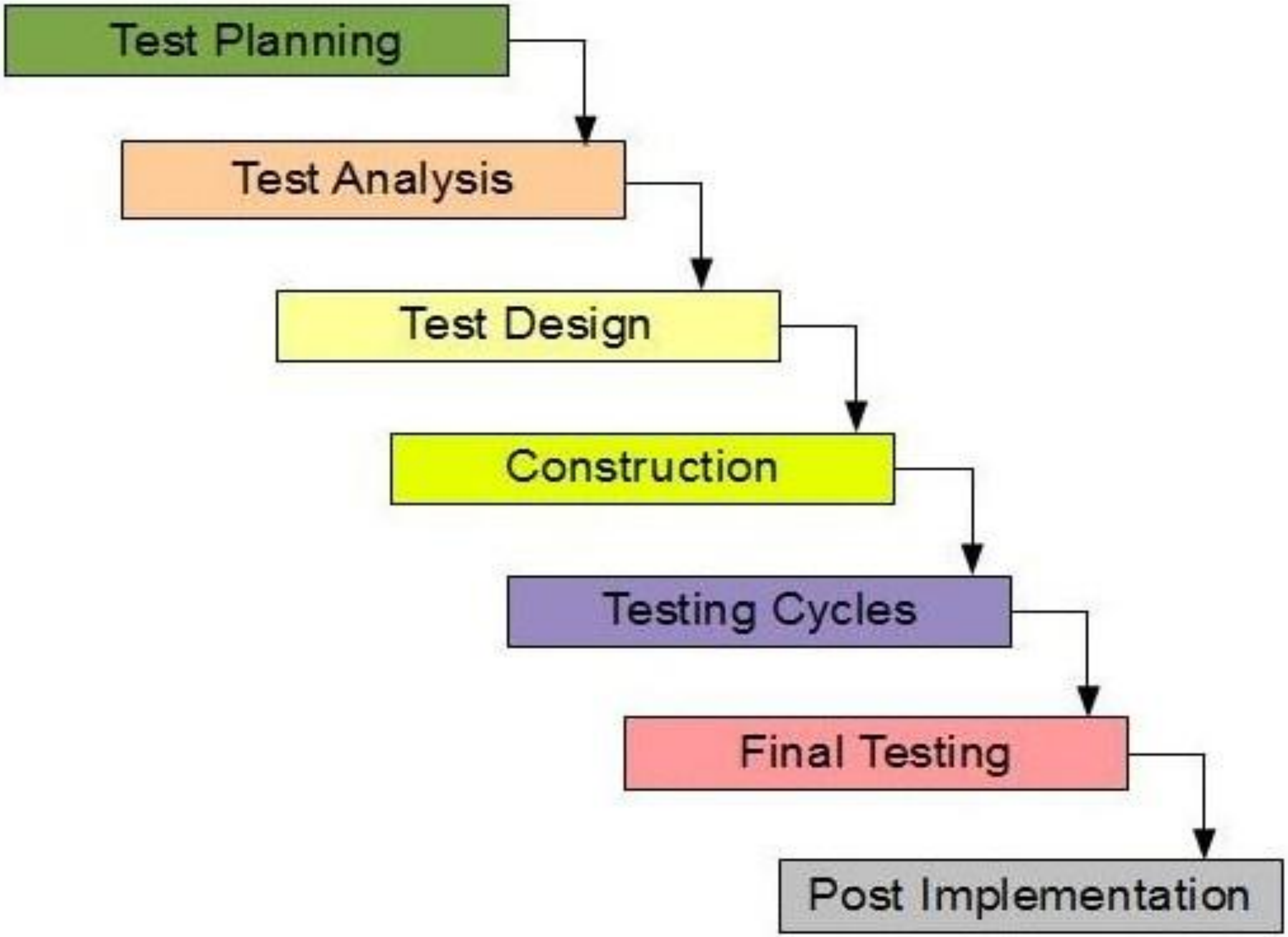
- **Comments:** Inadequate/ incorrect/ misleading or missing comments in the source code
- **Computational Error:** Improper computation of the formulae / improper business validations in code.
- **Data error:** Incorrect data population / update in database
- **Database Error:** Error in the database 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 completeDesign 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

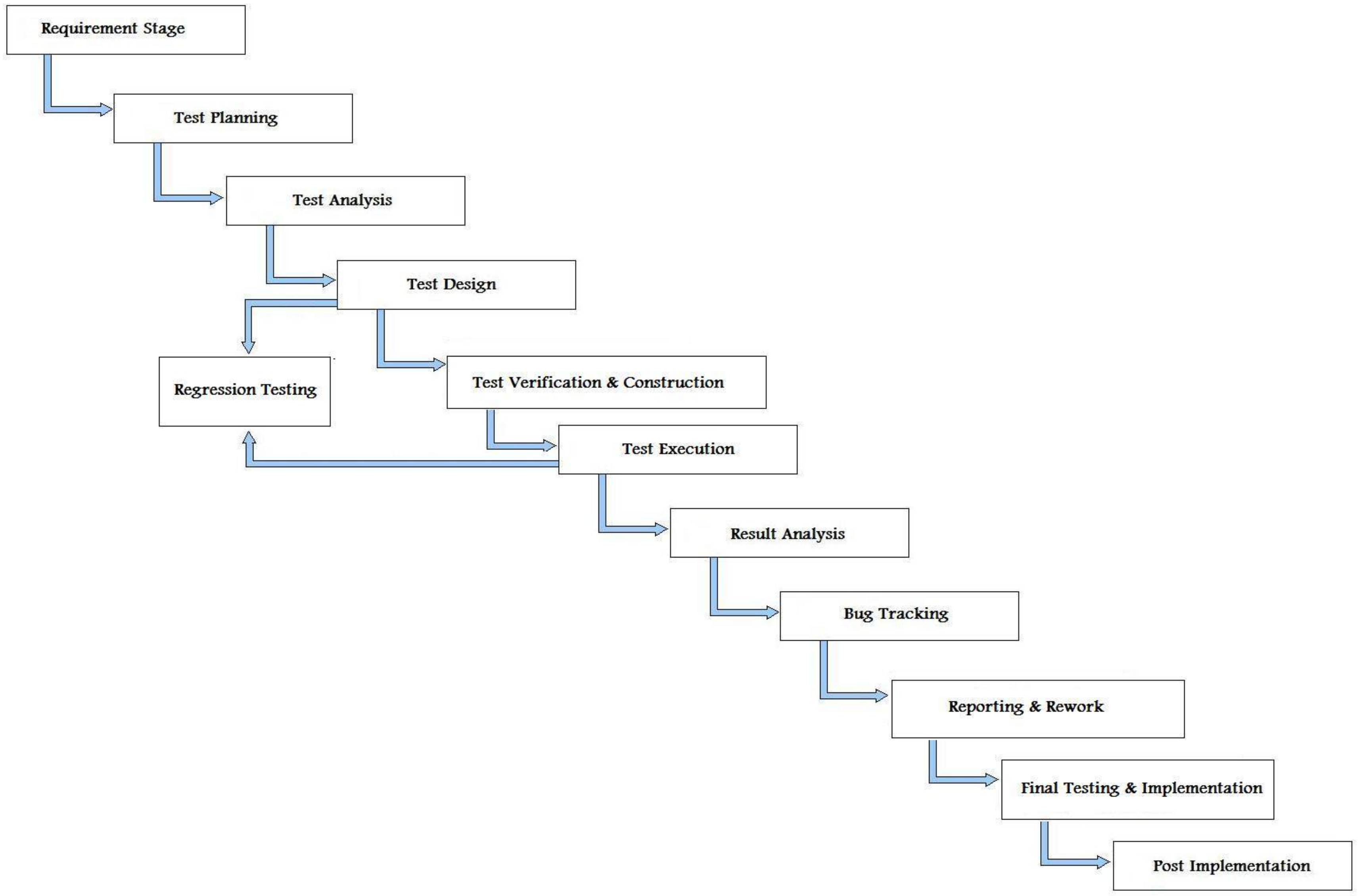
- **Message Error:** Inadequate/ incorrect/ misleading or missing error messages
- in source code
- **Navigation Error:** Navigation not coded correctly in source code
- **Performance Error:** An error related to performance/optimality of the code
- **Missing Requirements:** Implicit/Explicit requirements 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

Software Bug Report Template

- Name of Reporter:
- Email Id of Reporter:
- Version or Build: <*Version or Build of the product*>
- Module or component: <*mention here the name of tested module or component*>
- Platform / Operating System:
- Type of error: <*coding error / design error / suggestion / UI / documentation / text error / hardware error* >
- Priority:
- Severity:
- Status:
- Assigned to:
- Summary:
- Description: <*mention here the steps to reproduce, expected result and actual result*>

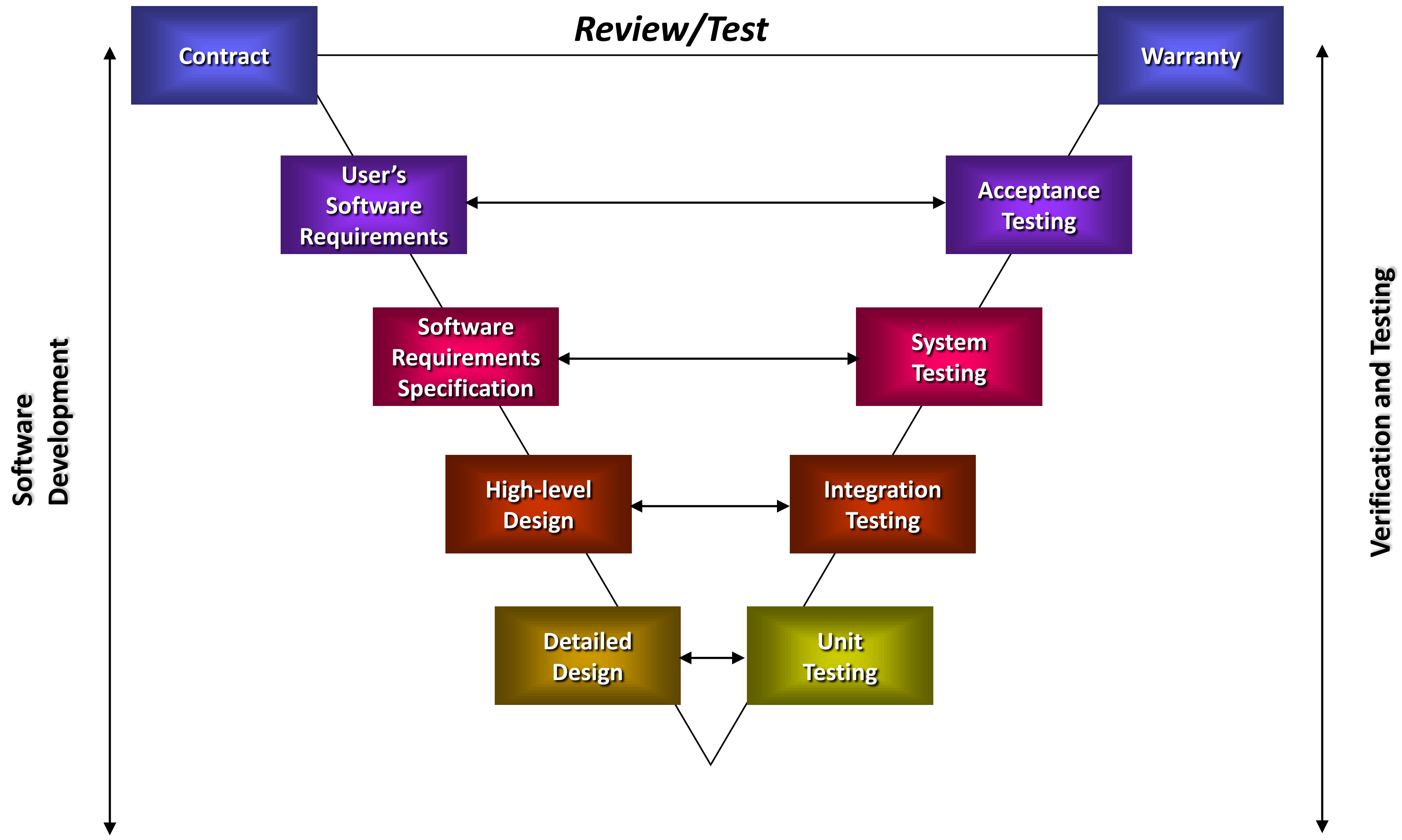
SOFTWARE TESTING LIFE CYCLE: PHASES


Test Planning
Test Analysis
Test Design
Construction
Testing Cycles
Final Testing
Post Implementation



Requirement Stage
Test Planning
Test Analysis
Test Design
Regression Testing
Test Verification & Construction
Test Execution
Result Analysis
Bug Tracking
Reporting & Rework
Final Testing & Implementation
Post Implementation

V - Model


Review/Test
Contract
Warranty
User's Software Requirements
Acceptance Testing
Software Requirements Specification
System Testing
High-level Design
Integration Testing
Detailed Design
Unit Testing
Software Development
Verification and Testing

V-model

- V- model means Verification and Validation model. Just like the **waterfall model**, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins.
- **V-Model** is one of the many software development models.
- Testing of the product is planned in parallel with a corresponding phase of development in **V-model**.

When to use the V-model:

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, test designing happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.
- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.