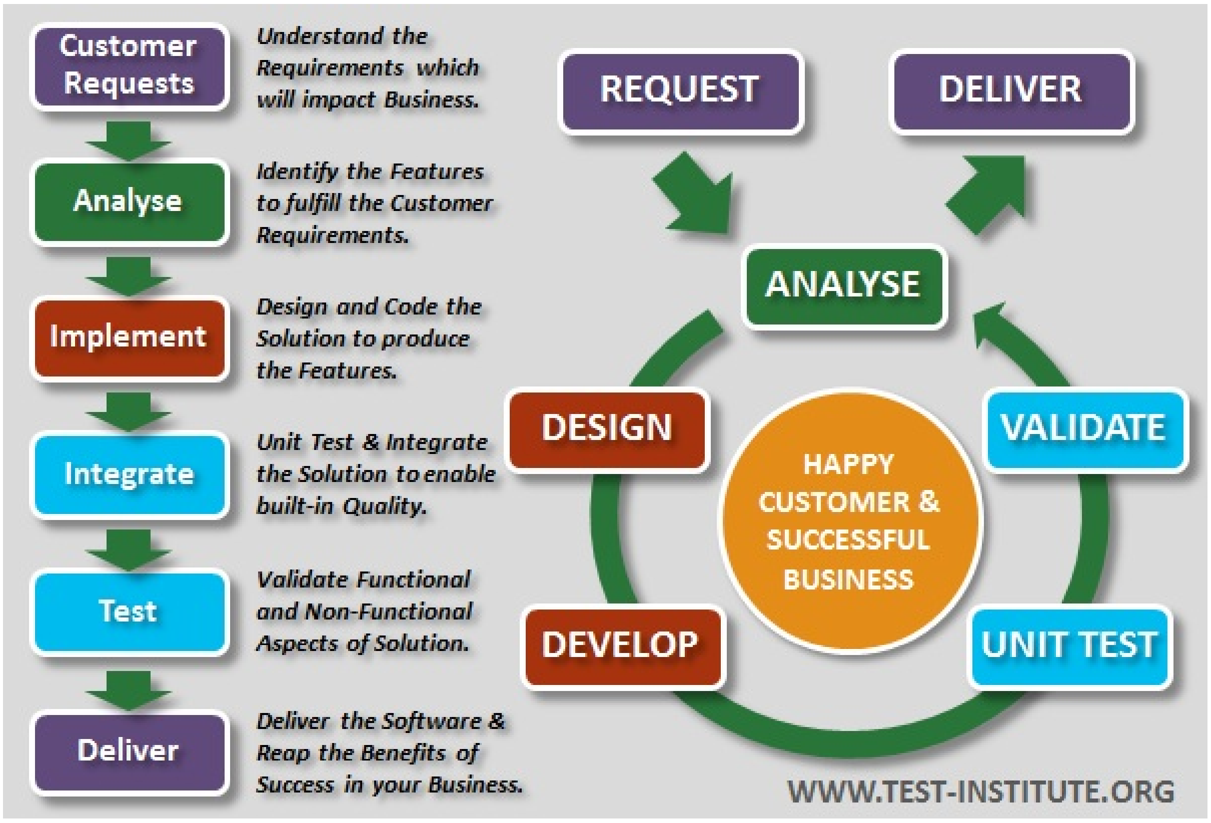
Software Testing Background

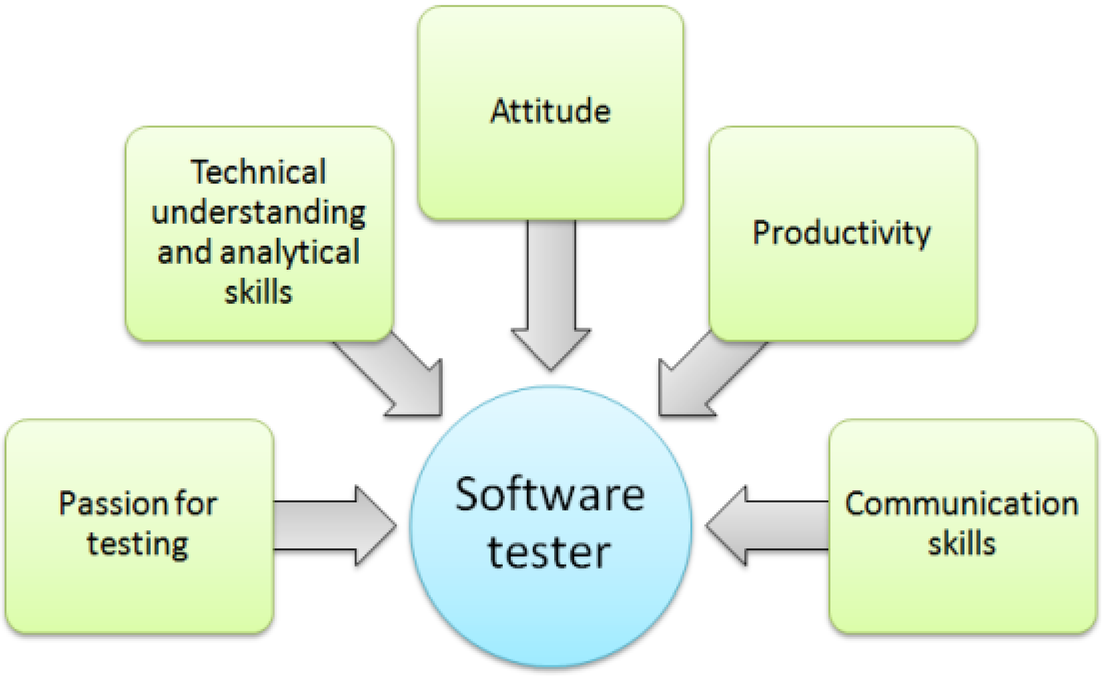
Introduction To Software Testing

Software testing is nothing but an art of investigating software to ensure that its quality under test is in line with the requirement of the client. Software testing is carried out in a systematic manner with the intent of finding defects in a system. It is required for evaluating the system. As the technology is advancing we see that everything is getting digitized. You can access your bank online, you can shop from the comfort of your home, and the options are endless. Have you ever wondered what would happen if these systems turn out to be defective?One small defect can cause a lot of financial loss. It is for this reason that software testing is now emerging as a very powerful field in IT.


Customer Requests
Understand the Requirements which will impact Business.
Analyse
Identify the Features to fulfill the Customer Requirements.
Implement
Design and Code the Solution to produce the Features.
Integrate
Unit Test & Integrate the Solution to enable built-in Quality.
Test
Validate Functional and Non-Functional Aspects of Solution.
Deliver
Deliver the Software & Reap the Benefits of Success in your Business.
REQUEST
DELIVER
ANALYSE
DESIGN
VALIDATE
HAPPY CUSTOMER & SUCCESSFUL BUSINESS
DEVELOP
UNIT TEST
WWW.TEST-INSTITUTE.ORG

Although like other products software never suffers from any kind of wear or tear or corrosion but yes, design errors can definitely make your life difficult if they go undetected. Regular testing ensures that the software is developed as per the requirement of the client. However, if the software is shipped with bugs embedded in it, you never know when they can create a problem and then it will be very difficult to rectify defect because scanning hundreds and thousands of lines of code and fixing a bug is not an easy task. You never know that while fixing one bug you may introduce another bug unknowingly in the system.

Software testing is now a very significant and integral part of software development. Ideally, it is best to introduce software testing in every phase of software development life cycle. Actually a majority of software development time is now spent on testing.



So, to summarize we can say that:

1. Software testing is required to check the reliability of the software
2. Software testing ensures that the system is free from any bug that can cause any kind of failure
3. Software testing ensures that the product is in line with the requirement of the client
4. It is required to make sure that the final product is user friendly
5. At the end software is developed by a team of human developers all having different viewpoints and approach. Even the smartest person has the tendency to make an error. It is not possible to create software with zero defects without incorporating software testing in the development cycle.
6. No matter how well the software design looks on paper, once the development starts and you start testing the product you will definitely find lots of defects in the design.

You cannot achieve software quality without software testing. Even if testers are not involved in actual coding they should work closely with developers to improve the quality of the code. For best results it is important that software testing and coding should go hand in hand.

I. **Software Testing Background**

1. Software is a set of instructions to perform some task.
2. Software is used in many applications of the real world.
3. Some of the examples are Application software, such as word processors,

firmware in an embedded system, middleware, which controls and co-ordinates distributed systems, system software such as operating systems, video games, and websites.
4. All of these applications need to run without any error and provide a quality service to the user of the application.
5. The software has to be tested for its accurate and correct working.

Software Testing: Testing can be defined in simple words as "Performing Verification and Validation of the Software Product" for its correctness and accuracy of working.
1. Software testing is as old as the hills in the history of digital computers.
2. The testing of software is an important means of assessing the software to determine its quality.
3. Since testing typically consumes 40% to50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering.
4. With the development of fourth generation languages (4GL), which speeds up the implementation process, the proportion of time devoted to testing increased.
5. As the amount of maintenance and upgrade of existing systems grow, significant amount of testing will also be needed to verify systems after changes are made.

II. **Software Error Case Studies** (Question: List the error case studies and explain any three :– list – 1 mark, explanation 3 marks = 4 marks)

The various error case studies related to software testing are

1. Disney Lion King

2. Intel Pentium Floating Point Division Bug

3. NASA Mars Polar Lander

4. Y2K Bug

The Disney Lion King case study identifies the problem that the software product which is developed should be compatible with the various configurations and the hardware compatibility should be tested on the various machines. The product developed by the company should be given for testing to various clients for testing.

The Intel Pentium floating point division bug occurs in the rare case when we try to develop high end mathematical project. Enter the following equation into your PC's calculator :(4195835 / 3145727) * 3145727 – 4195835. If it doesn't give the error its fine, but in few older machines it shows the floating point division error. But stakes the reputation of the company as it cannot solve the problem and monetary loss to the company.

The NASA Mars Polar Lander failed as each module that was designed for the launch was tested separately and functioned perfectly. But the various modules that were designed were never integrated and tested together. The switch installed to check the landing of the mars polar lander was tested separately and when integrated, the bit changed to 1 by the slight jerk caused all lander to fail.

The most common the Y2K bug was the recent one which caused lot of computers to give error as the year changed from 31.12.1999 to 1.1.2000 and the software’s which were developed to take the last two digits of the years turned to 00. The changes in the software’s were not possible as the developer who developed it either left the company and the development cost was too high.

III. **What is Bug**? (Question: Define Bug and list the terms related to software failure. :– definition - 1 mark, list – 1 mark, explanation – 2 marks = 4 marks)

Definition: A bug can be defined in simple term as any error or mistake that leads to the failure of the product or software either due to the specification problem or due to communication problem, regarding what is developed and what had to be developed.

1. Terms for software failure

The various terms related to software failure with respect to the area of application are listed as Defect, Variance, Fault, Failure, Problem, Inconsistency, Error, Feature, Incident, Bug, and Anomaly.

1. Problem, error, and bug are probably the most generic terms used.

2. Anomaly, incident, and variance don’t sound quite so negative and infer more unintended operation than an all-out failure.

3. Fault, failure, and defect tend to imply a condition that’s really severe, maybe even dangerous. It doesn’t sound right to call an incorrectly colored icon a fault. These words also tend to imply blame: “It’s his fault that the software failed.

4. As all the words sound the same they are distinguished based on the severity and the area in which the software failure has occurred.

5. When we run a program the error that we get during execution is termed on the basis of runtime error, compile time error, computational error, and assignment error.

6. The error can be removed by debugging, if not resolved leads to a problem and if the problem becomes large leads to software failure.

7. A bug can be defined as the initiation of error or a problem due to which fault, failure, incident or an anomaly occurs.

8. The program to find the factorial of a number given below lists few errors problem and failure in a program.

For example

```
1. #include<stdio.h>

2. void main()

3. {

4. int i , fact, n;

5. printf("enter the number ");

6. scanf("%d",&n);

7. for(i =1 ;i <=n;i++)

8. fact = fact * i;

9. printf ("the factorial of a number is "%d", fact);

10. }
```

- As in line number 4 the fact is not initialized to 1, so it takes garbage value and gives a wrong output, this is an example of a bug.
- If fact is initialized to zero (fact = 0) than the output will be zero as anything multiplied by zero will give the output as zero. This is a bug which can be removed by initializing fact = 1 during initializing.
- As the fact is declared as integer, for the number till 7! will work perfectly. When the number entered is 8, the output is garbage value as the integer limit is from – 32767 to +32768, so in declaration change the initialization to long int fact;

2. **Software Bug**:

A Formal Definition (Question: Write the factors that lead for bugs to occur in a project. :– definition - 1 mark, factors – 3 marks = 4 marks)

Definition: A bug can be defined in simple term as any error or mistake that leads to the failure of the product or software either due to the specification problem or due to communication problem, regarding what is developed and what had to be developed.

A software bug occurs when one or more of the following factors are true:

1. The software doesn't do something that the product specification says it should do.

2. The software does something that the product specification says it shouldn't do.

3. The software does something that the product specification doesn't mention.

4. The software doesn't do something that the product specification doesn't mention

but should.

5. The software is difficult to understand, hard to use, slow, or—in the software tester's eyes—will be viewed by the end user as just plain not right.

3. **Why do Bugs occur**? (Question: Why do bugs occur in a project? :– explanation – 3 marks, diagram 1 mark = 4 marks)

Bugs are a mismatch between expected result and actual result. Bugs play an important role in software testing.

In large projects bugs occur due to various reasons.

1. One of the extreme causes is the specification.

2. Specifications are the largest producer of bugs.

3. Either specifications are not written, specifications are not thorough enough, constantly changing or not communicated well to the development team.

4. Another bigger reason is that software is always created by human beings.

5. They know numerous things but are not expert and might make mistakes.

6. Further, there are deadlines to deliver the project on time. So increasing pressure and workload conduct in no time to check, compromise on quality and incomplete systems. So this leads to occurrence of Bugs.

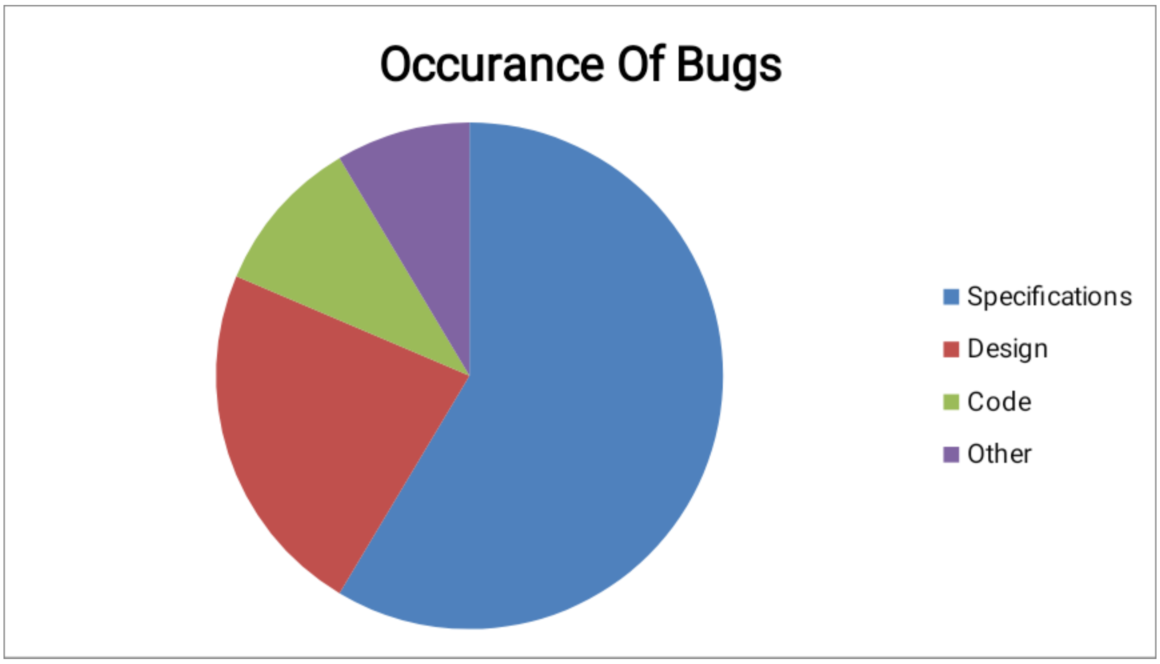


Figure 1.1

4. **What is the cost of bug in software testing**? (Question: What is the cost of bug in a project? – explanation - 3 marks, diagram - 1 mark = 4 marks)

1. If the error is made and the consequent defect is detected in the requirements phase then it is relatively cheap to fix it.

2. If an error is made and the consequent defect is found in the design phase or in construction phase then the design can be corrected and reissued with relatively little expense.

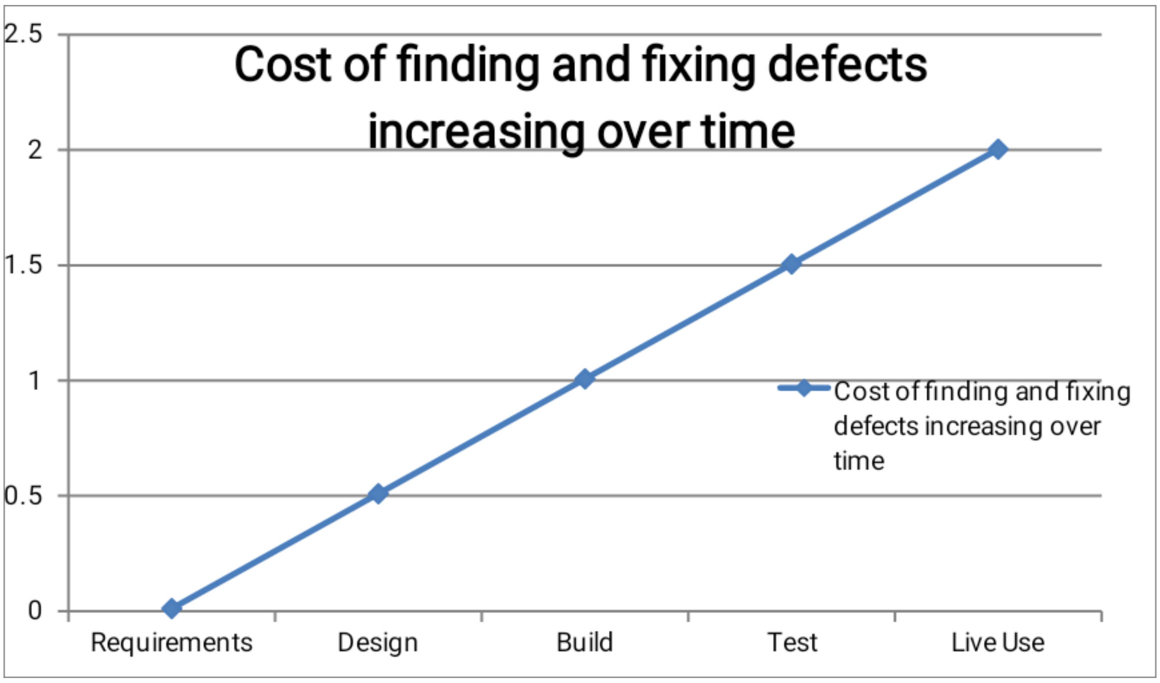


Figure 1.2

3. If a defect is introduced in the requirement specification and it is not detected until acceptance testing or even once the system has been implemented then it will be much more expensive to fix.

4. This is because rework will be needed in the specification and design before changes can be made in construction; because one defect in the requirements may well propagate into several places in the design and code.

5. All the testing work done-to that point will need to be repeated in order to reach the confidence level in the software that we require.

6. It is quite often the case that defects detected at a very late stage, depending on how serious they are, are not corrected because the cost of doing so is too expensive.

5. **What exactly does a Software Tester do**? (Question: What is a goal of software Tester in software development? :– explanation - 3 marks, example – 1 mark = 4 marks)

1. Software tester is the people who test the project module from the beginning of the project. 2. The major goal of the software testers is to find bud and to find them at the initial stages of the software development.

3. Once the bugs are found it is the responsibility of the software testers to give the

solution or at least the required solution for removing the bugs.

4. The goal of a software tester is to find bugs.

5. The goal of a software tester is to find bugs, and find them as early as possible.

6. The goal of a software tester is to find bugs, find them as early as possible, and make sure they get fixed.

For example

```
1. #include<stdio.h>
2. void main()
3. {
4. int i , fact, n;
5. printf("enter the number ");
6. scanf("%d",&n);
7. for(i =1 ;i <=n;i++)
8. fact = fact * i;
9. printf ("the factorial of a number is "%d", fact);
10. }
```

- As in line number 4 the fact is not initialized to 1, so it takes garbage value and gives a wrong output, this is an example of a bug.
- If fact is initialized to zero (fact = 0) than the output will be zero as anything multiplied by zero will give the output as zero. This is a bug which can be removed by initializing fact = 1 during initializing.
- As the fact is declared as integer, for the number till 7! will work perfectly. When the number entered is 8, the output is garbage value as the integer limit is from – 32767 to +32768, so in declaration change the initialization to long int fact;

6. **What makes a good Software Tester**? (Question: What are the traits of a good software tester? :– 4 marks any five points)

List of traits that most software testers should have:

1. They are explorers: - Software testers aren't afraid to venture into unknown situations. They love to get a new piece of software, install it on their PC, and

see what happens.

2. They are troubleshooters: - Software testers are good at figuring out why something doesn't work. As they find the bugs it is their responsibility to give the solution also.

3. They are relentless: - Software testers keep trying. They may see a bug that quickly vanishes or is difficult to re-create. Rather than dismiss it as a fluke, they will try every way possible to find it.

4. They are creative: - Testing the obvious isn't sufficient for software testers. Their job is to think up creative and even off-the-wall approaches to find bugs. Their perception of finding the bug is more than that of a programmer.

5. They are perfectionists: - They strive for perfection, but they know when it becomes unattainable and they want to be close as they can to the solution.

6. They exercise good judgment: - Software testers need to make decisions about what they will test, how long it will take, and if the problem they're looking at is really a bug. The job of a software tester is in time constraints. They have to find the bugs as early as possible.

7. They are tactful and diplomatic: - Software testers are always the bearers of bad news. They have to tell the programmers about the various bugs and the error that exist in their program and it has to be told carefully. Good software testers know how to do so tactfully and professionally and know how to work with programmers who aren't always tactful and diplomatic.

8. They are persuasive: - Bugs that testers find won't always be viewed as severe enough to be fixed. Testers need to be good at making their points clear, demonstrating why the bug does indeed need to be fixed, and following through on making it happen.

Software Development Lifecycle Models (Question: Explain the various Software Development Life Cycle (SDLC) models with advantages and disadvantages of waterfall model? : – list of 4 models – 4 marks, advantages – 1 mark, disadvantages 1 mark = 6 marks)

1. Big Bang Model

1. The big-bang model for software development shown in Figure 1.4 follows much the same principle.

2. A huge amount of matter (people and money) is put together, a lot of energy is expended—often violently—and out comes the perfect software product or it doesn't.

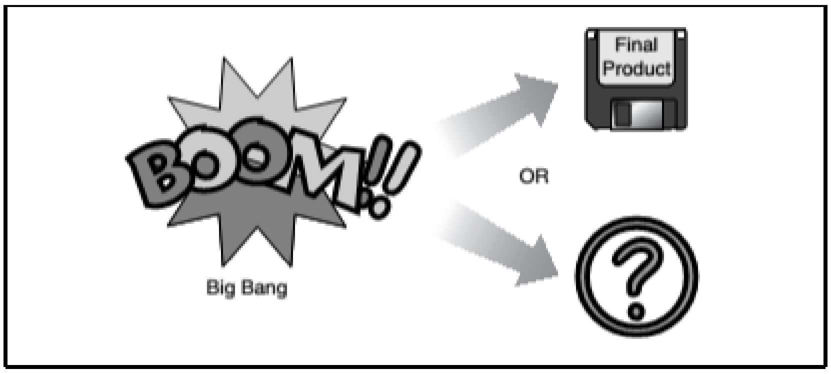


Figure 1.4

2. Code and Fix Model

1. The code-and-fix model shown in Figure 1.5 is usually the one that project teams fall into by default if they don't consciously attempt to use something else.

2. It's a step up, procedurally, from the big-bang model, in that it at least requires some idea of what the product requirements are.

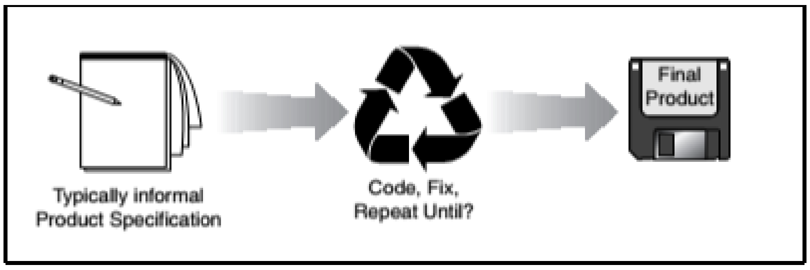


Figure 1.5

3. Waterfall Model (Question: Explain waterfall model with advantages and disadvantages of waterfall model? :– explanation – 2 marks, diagram – 2 marks, advantages – 1 mark, disadvantages – 1 marks = 6 marks)

1. The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall)

2. The phases are Requirement Gathering and Analysis, System Design, Implementation, Testing, Deployment of System and Maintenance.

Diagram of Waterfall-model:

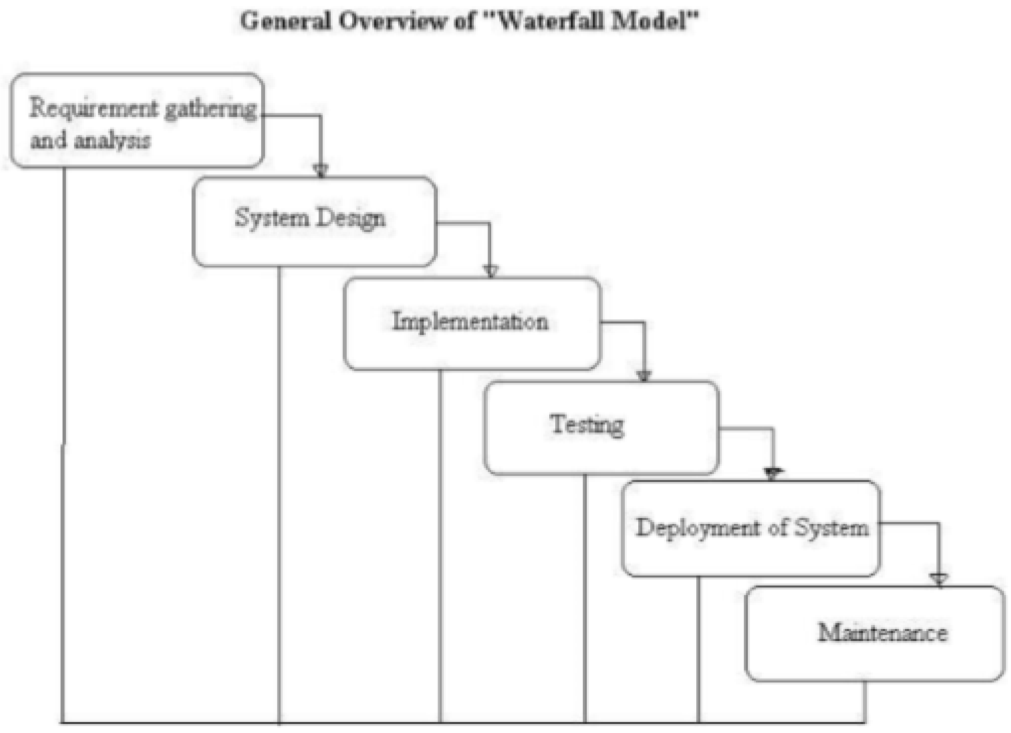

General Overview of "Waterfall Model"
Requirement gathering and analysis
System Design
Implementation
Testing
Deployment of System
Maintenance

Figure 1.6

Advantages of waterfall model:

1. Simple and easy to understand and use.

2. Easy to manage due to the rigidity of the model, each phase has specific deliverables and a review process.

3. Phases are processed and completed one at a time.

4. Works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

1. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.

2. No working software is produced until late during the life cycle.

3. High amounts of risk and uncertainty.

4. Not a good model for complex and object-oriented projects.

5. Poor model for long and on-going projects.

6. Not suitable for the projects where requirements are at a moderate to high risk of changing.

When to use the waterfall model:

1. Requirements are very well known, clear and fixed.

2. Product definition is stable.

3. Technology is understood.

4. There are no ambiguous requirements

5. Ample resources with required expertise are available freely

6. The project is short.

[illegible]

4. Spiral Model (Question: Explain spiral model with advantages and disadvantages of waterfall model? :– explanation – 2 marks, diagram – 2 marks, advantages – 1 mark, disadvantages – 1 marks = 6 marks)

1. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation.

2. A software project repeatedly passes through these phases in iterations.

3. The baseline spiral starting in the planning phase, requirements are gathered and risk is assessed.

4. The general idea behind the spiral model is that you don't define everything in detail at the very beginning.

5. Start small, define your important features, try them out, get feedback from your customers, and then move on to the next level.

6. Repeat this until you have your final product.

Diagram of Spiral Model:

•

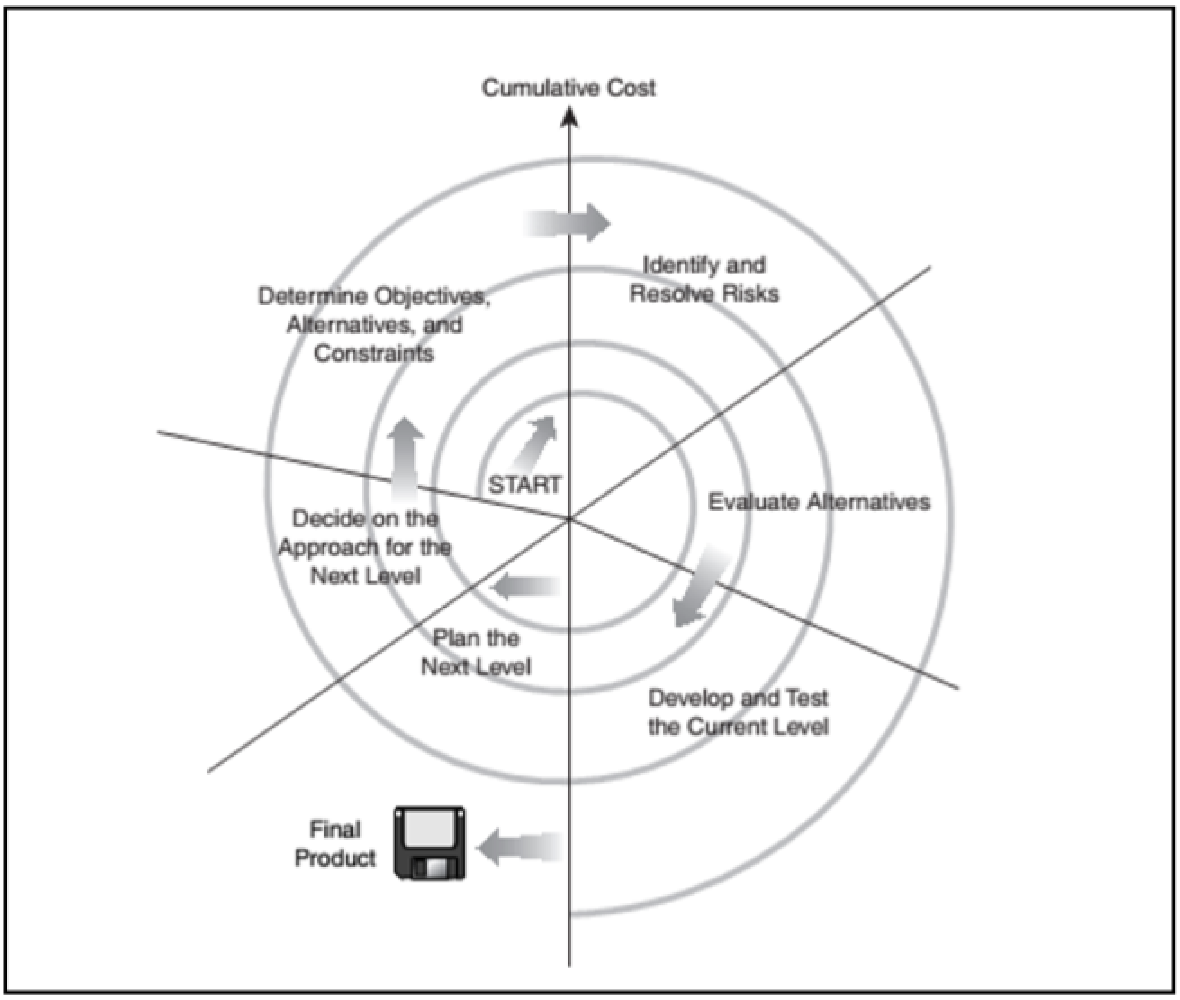

Cumulative Cost
Identify and Resolve Risks
Determine Objectives, Alternatives, and Constraints
START
Evaluate Alternatives
Decide on the Approach for the Next Level
Plan the Next Level
Develop and Test the Current Level
Final Product

Figure 1.7

Advantages of Spiral model:

1. High amount of risk analysis hence, avoidance of Risk is enhanced.
2. Good for large and mission-critical projects.
3. Strong approval and documentation control.
4. Additional Functionality can be added at a later date.
5. Software is produced early in the software life cycle.

Disadvantages of Spiral model:

1. Can be a costly model to use.
2. Risk analysis requires highly specific expertise.
3. Project's success is highly dependent on the risk analysis phase.

4. Doesn't work well for smaller projects.

When to use Spiral model:

1. When costs and risk evaluation is important

2. For medium to high-risk projects

3. Long-term project commitment unwise because of potential changes to economic priorities

4. Users are unsure of their needs

5. Requirements are complex

6. New product line

7. Significant changes are expected (research and exploration)

What is Software Testing?

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product.

Definition:

Software Testing Definition according to **ANSI/IEEE 1059** standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

Testing Axioms

(Question: Explain the various testing axioms of software testing? :– explain all 7 axioms = 6 marks)

Testing axioms are the rules of software testing and the knowledge that helps put some aspect of the overall process into perspective.

1. It's Impossible to Test a Program Completely As a new tester, you might believe that you can approach a piece of software, fully test it, find all the bugs, and assure that the software is perfect. Unfortunately, this isn't possible, even with the simplest programs, due to four key reasons:

1. The number of possible inputs is very large.

2. The number of possible outputs is very large.

3. The number of paths through the software is very large.

4. The software specification is subjective. You might say that a bug is in the eye of the beholder.

2. Software Testing Is a Risk-Based Exercise

1. If the tester decides to test every possibility of the product then it becomes impossible to deliver the product on time.

2. The testing has to be strictly done on the basis of the specifications provided by the client. 3. As the amount of testing increases with the time period the cost of the bug also increases. 4. Figure 1.8 shows the graph between the quality and the amount of testing that has to be done.

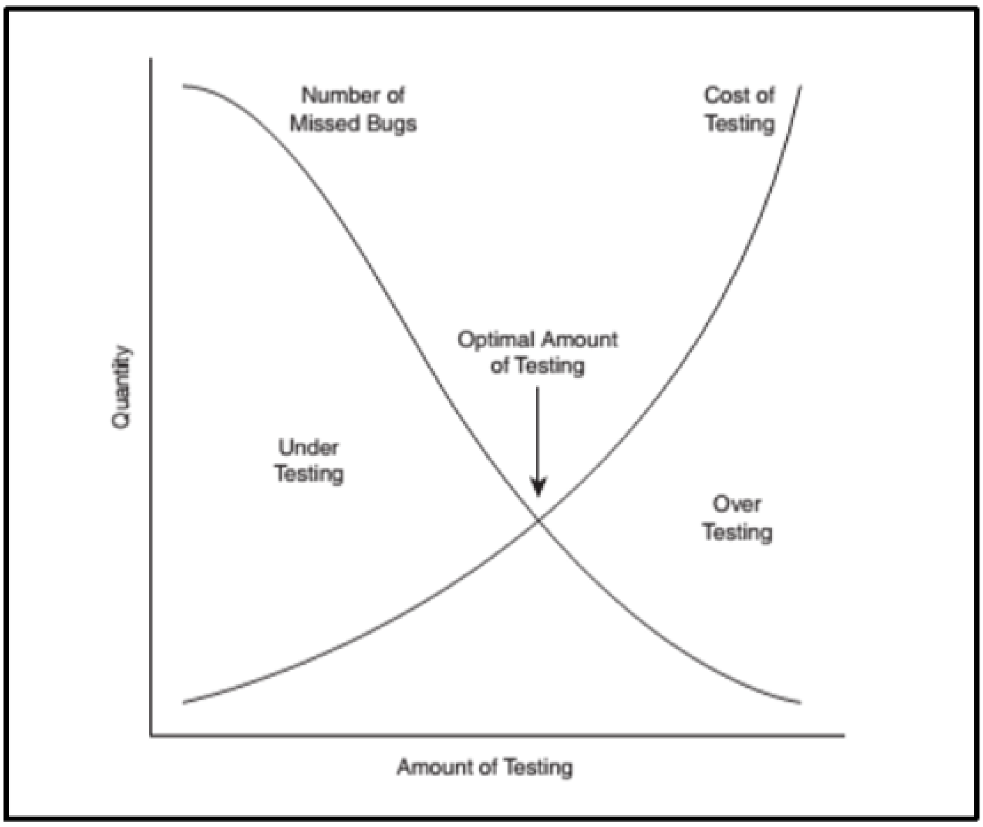


Figure 1.8

3. Testing Can't Show That Bugs Don't Exist Testing reports always show that the bugs exist. They can never show that the bugs don't exist. As a software tester to find bug at an early stage of the development is important and to give solution how it can be removed.

4. The More Bugs You Find, the More Bugs There Are There are even more similarities between real bugs and software bugs. Both types tend to come in

groups.

There are several reasons for this:

1. Programmers have bad days: - Like all of us, programmers can have off days. Code written one day may be perfect; code written another may be sloppy. One bug can be a tell-tale sign that there are more nearby.

2. Programmers often make the same mistake: - Everyone has habits. A programmer who is prone to a certain error will often repeat it.

3. Some bugs are really just the tip of the iceberg: - Very often the software's design or architecture has a fundamental problem. A tester will find several bugs that at first may seem unrelated but eventually are discovered to have one primary serious cause.

5. The Pesticide Paradox

1. The term pesticide paradox can be described as the phenomenon that the more you test software, the more immune it becomes to your tests.

2. The same thing happens to insects with pesticides (see Figure 1.9).

3. If you keep applying the same pesticide, the insects eventually build up resistance and the pesticide no longer works.

4. Hence it is always advised to test the software with the difference set of inputs and parameters.

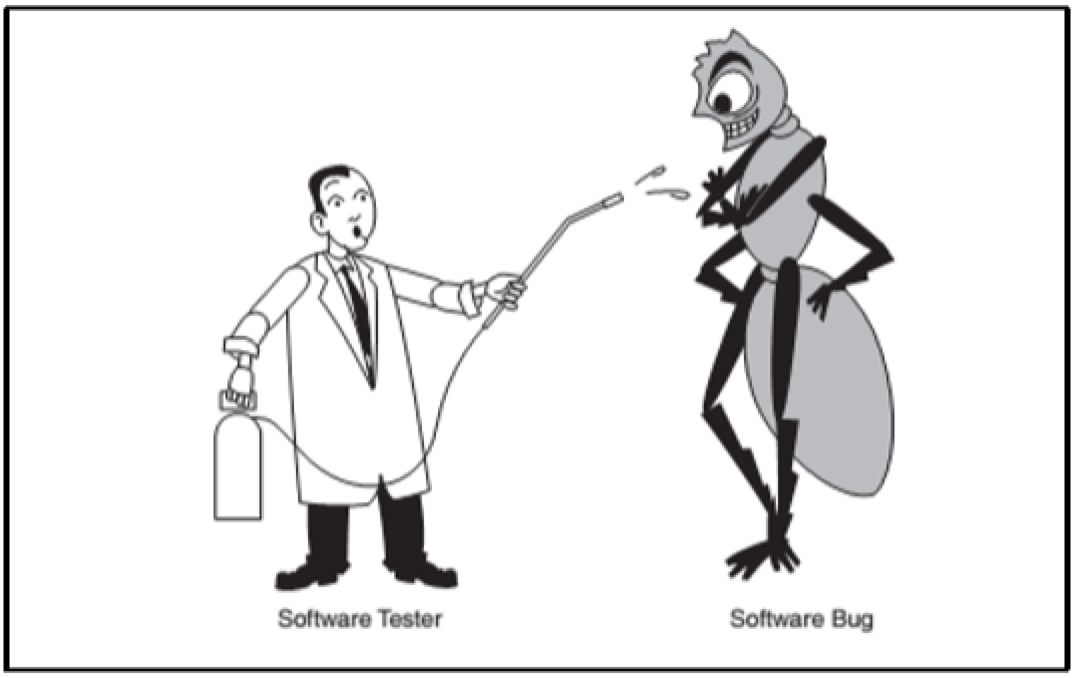


Figure 1.9

6. Not All the Bugs You Find Will Be Fixed There are several reasons why you might choose not to fix a bug:

1. There's not enough time: - In every project there are always too many software features, too few people to code and test them, and not enough room left in the schedule to finish. If you're working on a tax preparation program, April 15 isn't going to move—you must have your software ready in time.

2. It's really not a bug: -Maybe you've heard the phrase, "It's not a bug, it's a feature!" It's not uncommon for misunderstandings, test errors, or spec changes to result in would-be bugs being dismissed as features.

3. It's too risky to fix: -Unfortunately, this is all too often true. Software is fragile, intertwined, and sometimes like spaghetti. You might make a bug fix that causes other bugs to appear. Under the pressure to release a product under a tight schedule, it might be too risky to change the software. It may be better to leave in the known bug to avoid the risk of creating new, unknown ones.

4. It's just not worth it: - This may sound harsh, but it's reality. Bugs that would occur infrequently or bugs that appear in little-used features may be dismissed. Bugs that have work-around, ways that a user can prevent or avoid the bug, are often not fixed. It all comes down to a business decision based on risk.

7. When a Bug's a Bug Is Difficult to Say

1. The software doesn't do something that the product specification says it should do. 2. The software does something that the product specification says it shouldn't do.

3. The software does something that the product specification doesn't mention.

4. The software doesn't do something that the product specification doesn't mention but should.

5. The software is difficult to understand, hard to use, slow, or—in the

software tester's eyes—will be viewed by the end user as just plain not right.

What is Software Testing Life Cycle (STLC)?

Software Testing Life Cycle (STLC) is defined as a sequence of activities conducted to perform Software Testing.

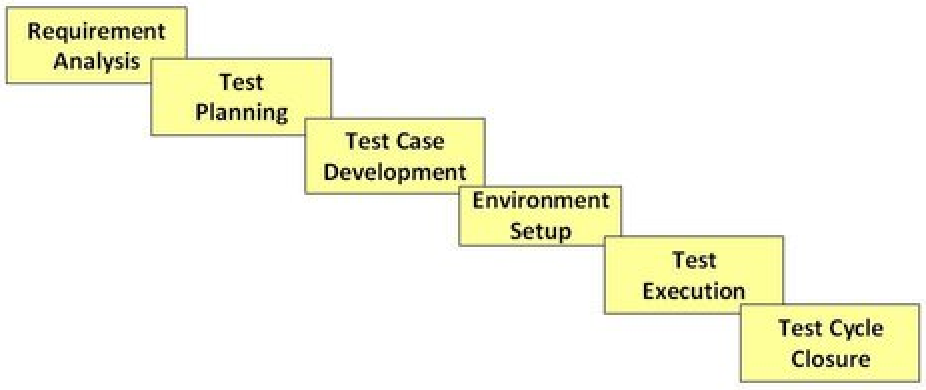
Contrary to popular belief, Software Testing is not a just a single activity. It consists of a series of activities carried out methodologically to help certify your software product.

Below are the phases of STLC:

- Requirement Analysis
- Test Planning
- Test case development
- Test Environment setup
- Test Execution
- Test Cycle closure

Each of these stages has a definite Entry and Exit criteria, Activities & Deliverables associated with it.

Different Phases of the STLC Model



STLC Diagram

What is Entry and Exit Criteria?

- **Entry Criteria:** Entry Criteria gives the prerequisite items that must be completed before testing can begin.
- **Exit Criteria:** Exit Criteria defines the items that must be completed before testing can be concluded

You have Entry and Exit Criteria for all levels in the Software Testing Life Cycle (STLC)

In an Ideal world, you will not enter the next stage until the exit criteria for the previous stage is met. But practically this is not always possible. So for this tutorial, we will focus on activities and deliverables for the different stages in STLC life cycle. Let's look into them in detail.

Requirement Analysis

During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.

The QA team may interact with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) to understand the requirements in detail.

Requirements could be either Functional (defining what the software must do) or Non Functional (defining system performance /security availability)

Automation feasibility for the given testing project is also done in this stage.

Activities

- Identify types of tests to be performed.
- Gather details about testing priorities and focus.
- Prepare Requirement Traceability Matrix (RTM).
- Identify test environment details where testing is supposed to be carried out.
- Automation feasibility analysis (if required).

Deliverables

- RTM
- Automation feasibility report. (if applicable)

Test Planning

Typically, in this stage, a Senior QA manager will determine effort and cost estimates for the project and would prepare and finalize the Test Plan. In this phase, Test Strategy is also determined.

Activities

- Preparation of test plan/strategy document for various types of testing
- Test tool selection
- Test effort estimation
- Resource planning and determining roles and responsibilities.
- Training requirement

Deliverables

- Test plan /strategy document.
- Effort estimation document.

Test Case Development

This phase involves the creation, verification and rework of test cases & test scripts. Test data, is identified/created and is reviewed and then reworked as well.

Activities

- Create test cases, automation scripts (if applicable)
- Review and baseline test cases and scripts
- Create test data (If Test Environment is available)

Deliverables

- Test cases/scripts
- Test data

Test Environment Setup

Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process and ***can be done in parallel with Test Case Development Stage***. ***Test team may not be involved in this activity*** if the customer/development team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment.

Activities

- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data
- Perform smoke test on the build

Deliverables

- Environment ready with test data set up
- Smoke Test Results.

Test Execution

During this phase, the testers will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

Activities

- Execute tests as per plan
- Document test results, and log defects for failed cases
- Map defects to test cases in RTM
- Retest the Defect fixes

- Track the defects to closure

Deliverables

- Completed RTM with the execution status
- Test cases updated with results
- Defect reports

Test Cycle Closure

Testing team will meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in the future, taking lessons from the current test cycle. The idea is to remove the process bottlenecks for future test cycles and share best practices for any similar projects in the future.

Activities

- Evaluate cycle completion criteria based on Time, Test coverage, Cost,Software, Critical Business Objectives, Quality
- Prepare test metrics based on the above parameters.
- Document the learning out of the project
- Prepare Test closure report
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.

Deliverables

- Test Closure report
- Test metrics

Software Testing Terms: Precision, and Accuracy

1. **Precision and Accuracy** (Question: Explain with diagram the difference between precision and accuracy. :- diagram – 1 mark, explanation – 3 marks = 4 marks)

1. The Accuracy of a measurement system is the degree of closeness of measurements of a quantity to that quantity's actual (true) value.

2. The Precision of a measurement system, also called reproducibility or repeatability, is the degree to which repeated measurements under unchanged conditions show the same results.

3. Although the two words precision and accuracy can be synonymous in

colloquial use, they are deliberately contrasted in the context of the scientific method.

Diagram for Accuracy and Precision

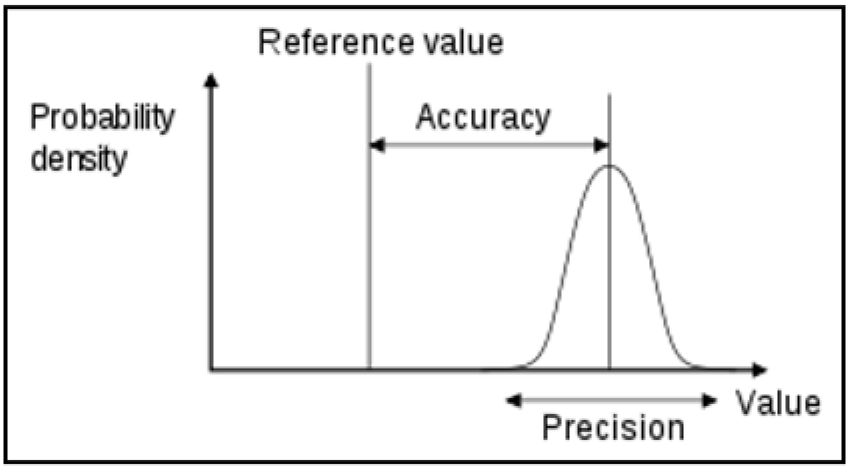


Figure 1.10

For example –

1. A measurement system can be accurate but not precise, precise but not accurate, neither, or both.

2. If an experiment contains a systematic error, then increasing the sample inputs generally increases precision but does not improve accuracy.

3. The result would be a consistent yet inaccurate string of results from the flawed software.

4. Eliminating the systematic error improves accuracy but does not change precision.

2. **Verification and Validation** (Question: Explain the concept of verification and validation in software testing. :- 2 marks + 2 marks = 4 marks)

Verification

1. It makes sure that the product is designed to deliver all functionality to the customer.

2. Verification is done at the starting of the development process. It includes

reviews and meetings, walkthroughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications.

3. It answers the questions like: Am I building the product right?

4. Am I accessing the data right (in the right place; in the right way).

5. It is a Low level activity

6. Performed during development on key art facts, like walkthroughs, reviews and inspections, mentor feedback, training, checklists and standards.

7. Demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle.

Validation

1. Determining if the system complies with the requirements and performs functions for which it is intended and meets the organization's goals and user needs.

2. Validation is done at the end of the development process and takes place after verifications are completed.

3. It answers the question like: Am I building the right product?

4. Am I accessing the right data (in terms of the data required to satisfy the requirement).

Quality and Reliability

They sound similar. And they are often used (erroneously) interchangeably. Both terms, reliable and quality, can be used to describe a software application that has a low degree of error. But there is one fundamental difference between the two.

Software Reliability is objective, measurable, and can be estimated, whereas Software Quality is based on primarily subjective criteria.

When looking at software **quality**, Dr. Scott Pressman, in his book *Software Engineering: A Practitioner's Approach*, defines it as the following:

In the context of software engineering, software quality measures how well software is designed (quality of design), and how well the software conforms to that design (quality of conformance).
The term "measures" in this definition is slightly misleading in my opinion, since many if not all of the aspects of software quality are entirely subjective. Some of the factors that go into determining software quality are:

- Understand ability - Do variables have descriptive names? Are methods commented so that their purpose is clear? Are any deviations in logic flow

adequately commented?
- Conciseness - Is all code reachable? Is any code redundant? How many statements within loops could be placed outside the loop, thus reducing computation time? Are branch decisions too complex?
- Portability - Does the program depend upon system or library routines unique to a particular installation? Have machine-dependent statements been flagged and commented?
- Maintainability - Have design patterns that lend themselves to ease of maintainability been employed, (i.e. Code to Interface)?

Any measurements of these factors are purely qualitative, and therefore subjective. In Maintainability, for example, one software engineer might feel that Coding to Interface promotes easy of future maintenance, while another may feel that abstract classes with only one implementation add complexity without any additional benefit.

Reliability, on the other hand, is defined by ANSI as "the probability of failure-free operation of a computer program in a specified environment for a specified time." It is arguable that Dr. Pressman's definition can be applied to reliability as well, and this would be the "quality of conformance" half of the equation. This would mean that reliability is simply another factor of quality. However, I tend to separate the two, for one very important reason. Because **reliability can be measured with quantitative metrics**, two benefits can be realized. 1) The amount of effort needed to achieve a certain threshold of reliability can be estimated in the planning phase. And 2) the reliability of a released application can be objectively measured.

Bottom line, because quality is subjective, there is no way to compare the quality of two software applications, or more importantly, set minimum quality standards for a piece of software. But because measures of reliability are objective and quantitative, you can.

Quality Assurance and Quality Control

(Question: Explain the concept of Quality Assurance and Quality control in software testing. :- 2 marks + 2 marks =4 marks)

Quality Assurance

1. A part of quality management focused on providing confidence that quality requirements will be fulfilled.

2. All the planned and systematic activities implemented within the quality system that can be demonstrated to provide confidence that a product or service will fulfill requirements for quality

3. Quality Assurance is fundamentally focused on planning and documenting those processes to assure quality including things such as quality plans and inspection and test plans.

4. Quality Assurance is a system for evaluating performance, service, of the quality of a product against a system, standard or specified requirement for customers.

5. Quality Assurance is a complete system to assure the quality of products or services. It is not only a process, but a complete system including also control. It is a way of management.

Quality Control

1. A part of quality management focused on fulfilling quality requirements.

2. The operational techniques and activities used to fulfil requirements for quality.

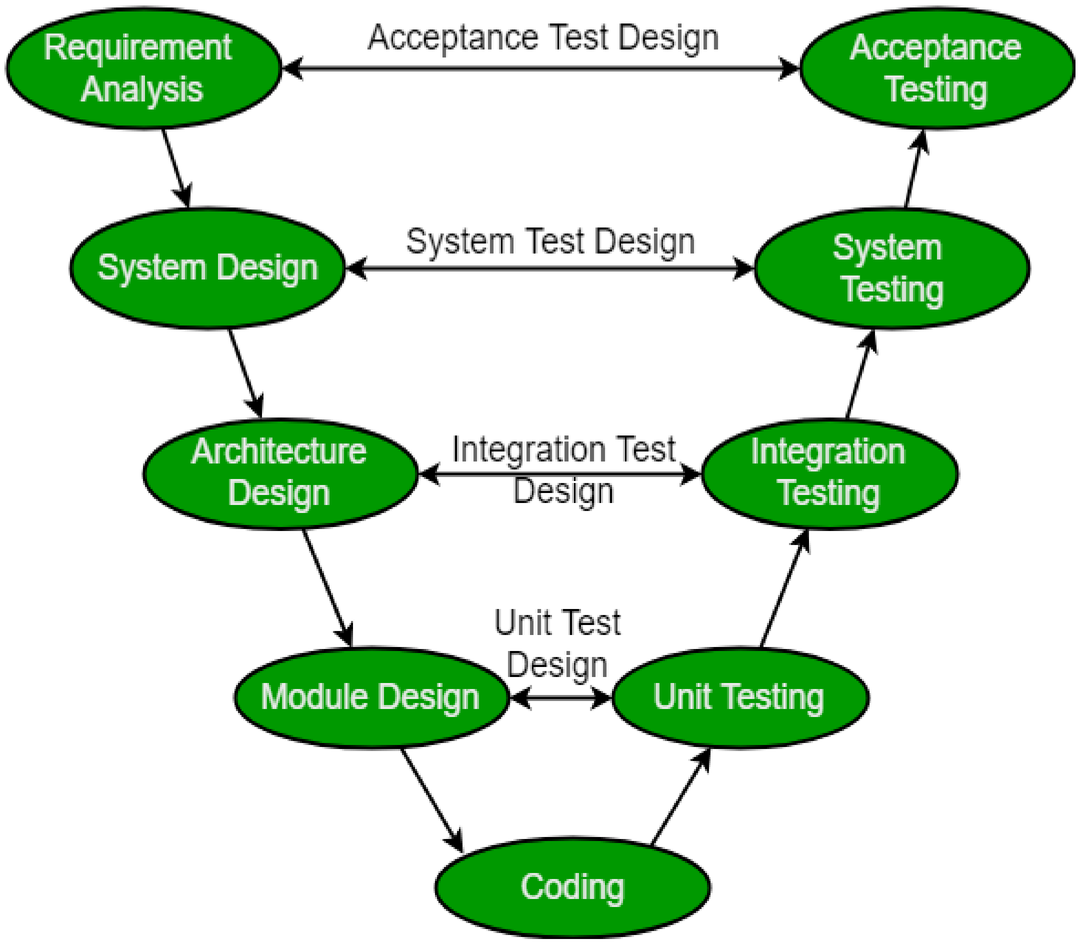
3. Quality Control on the other hand is the physical verification that the product conforms to these planned arrangements by inspection, measurement etc.

4. Quality Control is the process involved within the system to ensure job management, competence and performance during the manufacturing of the product or service to ensure it meets the quality plan as designed.

5. Quality Control just measures and determines the quality level of products or services. It is a process itself.

Software Testing| SDLC V-Model

The V-model is a type of SDLC model where process executes in a sequential manner in V-shape. It is also known as Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage. Development of each step directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e. for each development activity, there is a testing activity corresponding to it.



Verification: It involves static analysis technique (review) done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements meet.

Validation: It involves dynamic analysis technique (functional, non-functional), testing done by executing code. Validation is the process to evaluate the software after the completion of the development phase to determine whether software meets the customer expectations and requirements.
So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation phases are joined by coding phase in V-shape. Thus it is called V-Model.

Design Phase:

- **Requirement Analysis:** This phase contains detailed communication with the customer to understand their requirements and expectations. This stage is known as Requirement Gathering.
- **System Design:** This phase contains the system design and the complete hardware and communication setup for developing product.
- **Architectural Design:** System design is broken down further into modules taking up different functionalities. The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood.
- **Module Design:** In this phase the system breaks dowm into small modules. The

detailed design of modules is specified, also known as Low-Level Design (LLD).

Testing Phases:

- **Unit Testing:** Unit Test Plans are developed during module design phase. These Unit Test Plans are executed to eliminate bugs at code or unit level.
- **Integration testing:** After completion of unit testing Integration testing is performed. In integration testing, the modules are integrated and the system is tested. Integration testing is performed on the Architecture design phase. This test verifies the communication of modules among themselves.
- **System Testing:** System testing test the complete application with its functionality, inter dependency, and communication.It tests the functional and non-functional requirements of the developed application.
- **User Acceptance Testing (UAT):** UAT is performed in a user environment that resembles the production environment. UAT verifies that the delivered system meets user's requirement and system is ready for use in real world.

Industrial Challange: As the industry has evolved, the technologies have become more complex, increasingly faster, and forever changing, however, there remains a set of basic principles and concepts that are as applicable today as when IT was in its infancy.

- Accurately define and refine user requirements.
- Design and build an application according to the authorized user requirements.
- Validate that the application they had built adhered to the authorized business requirements.

Principles of V-Model:

- **Large to Small:** In V-Model, testing is done in a hierarchical perspective, For example, requirements identified by the project team, create High-Level Design, and Detailed Design phases of the project. As each of these phases is completed the requirements, they are defining become more and more refined and detailed.
- **Data/Process Integrity:** This principle states that the successful design of any project requires the incorporation and cohesion of both data and processes. Process elements must be identified at each and every requirements.
- **Scalability:** This principle states that the V-Model concept has the flexibility to accommodate any IT project irrespective of its size, complexity or duration.
- **Cross Referencing:** Direct correlation between requirements and corresponding testing activity is known as cross-referencing.
- **Tangible Documentation:** This principle states that every project needs to create a document. This documentation is required and applied by both the project development team and the support team. Documentation is used to maintaining the application once it is available in a production environment.

Why preferred?

- It is easy to manage due to the rigidity of the model. Each phase of V-Model has specific deliverables and a review process.
- Proactive defect tracking – that is defects are found at early stage.

When to use?

- Where requirements are clearly defined and fixed.
- The V-Model is used when ample technical resources are available with technical expertise.

Advantages:

- This is a highly disciplined model and Phases are completed one at a time.

- V-Model is used for small projects where project requirements are clear.
- Simple and easy to understand and use.
- This model focuses on verification and validation activities early in the life cycle thereby enhancing the probability of building an error-free and good quality product.
- It enables project management to track progress accurately.

Disadvantages:

- High risk and uncertainty.
- It is not a good for complex and object-oriented projects.
- It is not suitable for projects where requirements are not clear and contains high risk of changing.
- This model does not support iteration of phases.
- It does not easily handle concurrent events.