Software Development Life Cycle (SDLC)

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

Need of SDLC

The development team must determine a suitable life cycle model for a particular plan and then observe to it.

Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representative about when and what to do. Otherwise, it would point to chaos and project failure. This problem can be defined by using an example. Suppose a software development issue is divided into various parts and the parts are assigned to the team members. From then on, suppose the team representative is allowed the freedom to develop the roles assigned to them in whatever way they like. It is possible that one representative might start writing the code for his part, another might choose to prepare the test documents first, and some other engineer might begin with the design phase of the roles assigned to him. This would be one of the perfect methods for project failure.

A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.

SDLC Cycle

SDLC Cycle represents the process of developing software. SDLC framework includes the following steps:

The stages of SDLC are as follows:

Stage1: Planning and requirement analysis

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Stage2: Defining Requirements

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

Stage3: Designing the Software

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

Stage4: Developing the project

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

Stage5: Testing

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.

During this stage, unit testing, integration testing, system testing, acceptance testing are done.

Stage6: Deployment

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

Stage7: Maintenance

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.

Objectives of Software Testing

- The main goal of software testing is to find bugs as early as possible and fix bugs and make sure that the software is bug-free.

  Important Goals of Software Testing:

  - Detecting bugs as soon as feasible in any situation.
  - Avoiding errors in a project's and product's final versions.
  - Inspect to see whether the customer requirements criterion has been satisfied.
  - Last but not least, the primary purpose of testing is to gauge the project and product level of quality.

  The goals of software testing may be classified into three major categories as follows:

  1. **Immediate Goals**
  2. **Long-term Goals**
  3. **Post-Implementation Goals**

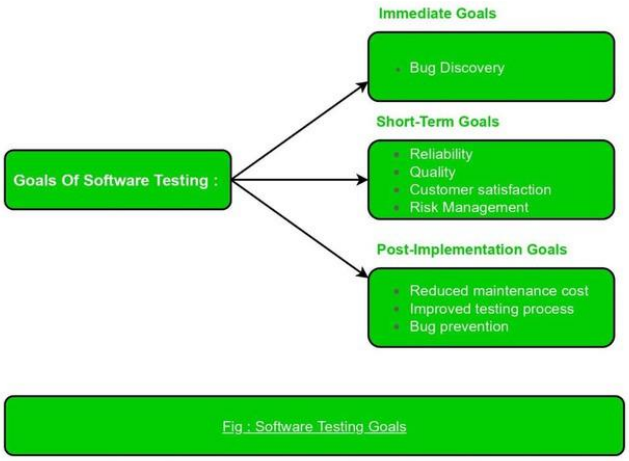

Fig : Software Testing Goals

1. Immediate Goals: These objectives are the direct outcomes of testing. These objectives may be set at any time during the SDLC process. Some of these are covered in detail below:

- **Bug Discovery:** This is the immediate goal of software testing to find errors at any stage of software development. The number of bugs is discovered in the early stage of testing. The primary purpose of software testing is to detect flaws at any step of the development process. The higher the number of issues detected at an early stage, the higher the software testing success rate.

- **Bug Prevention:** This is the immediate action of bug discovery, that occurs as a result of bug discovery. Everyone in the software development team learns how to code from the behavior and analysis of issues detected, ensuring that bugs are not duplicated in subsequent phases or future projects.

2. Long-Term Goals: These objectives have an impact on product quality in the long run after one cycle of the SDLC is completed. Some of these are covered in detail below:

- **Quality:** This goal enhances the quality of the software product. Because software is also a product, the user's priority is its quality. Superior quality is ensured by thorough testing. Correctness, integrity, efficiency, and reliability are all aspects that influence quality. To attain quality, you must achieve all of the above-mentioned quality characteristics.
- **Customer Satisfaction:** This goal verifies the customer's satisfaction with a developed software product. The primary purpose of software testing, from the user's standpoint, is customer satisfaction. Testing should be extensive and thorough if we want the client and customer to be happy with the software product.
- **Reliability:** It is a matter of confidence that the software will not fail. In short, reliability means gaining the confidence of the customers by providing them with a quality product.

- **Risk Management:** Risk is the probability of occurrence of uncertain events in the organization and the potential loss that could result in negative consequences. Risk management must be done to reduce the failure of the product and to manage risk in different situations.

3. Post-Implemented Goals: After the product is released, these objectives become critical. Some of these are covered in detail below:

- **Reduce Maintenance Cost:** Post-released errors are costlier to fix and difficult to identify. Because effective software does not wear out, the maintenance cost of any software product is not the same as the physical cost. The failure of a software product due to faults is the only expense of maintenance. Because they are difficult to discover, post-release mistakes always cost more to rectify. As a result, if testing is done thoroughly and effectively, the risk of failure is lowered, and maintenance costs are reduced as a result.
- **Improved Software Testing Process:** These goals improve the testing process for future use or software projects. These goals are known as post-implementation goals. A project's testing procedure may not be completely successful, and there may be room for improvement. As a result, the bug history and post-implementation results can be evaluated to identify stumbling blocks in the current testing process that can be avoided in future projects.

Limitations of Testing in Software Development

There are certain limitations of testing in software testing that need to be considered such as incomplete test coverage, the presence of undetectable errors, time and resource constraints, reliance on test data, and the inability to guarantee absolute correctness. It's important to acknowledge these limitations and implement strategies to mitigate their impact during the testing process.

Key Limitations of Testing in Software Development:

- **Testing cannot typically uncover things that are missing or unknown**:

Tests can be written to find and identify known issues. Still, if there are issues that aren't well understood, there's no way to design an appropriate test to find them. Therefore, testing simply cannot guarantee that an application is free from errors.

- **It's impossible to test for all conditions**:

There is effectively an infinite number and variety of inputs that can go into an application, and testing them all, even if theoretically possible, is never going to be practical. Testing must cover the known probable inputs, and this is a balance between quality and deadlines.

- **Testing usually gives no insight into the root causes of errors**:

These causes need to be identified to prevent repeating the same mistakes in the future. Yet, testing will only tell whether or not the issues are present.

Other limitations:

- **Time and resource constraints:**

Limited time and resources may restrict the extent of testing that can be performed.

- **Reliance on test data:**

Testing relies on the availability of representative and comprehensive test data for accurate results.

- **Inability to guarantee absolute correctness:**

Testing can provide confidence in the software's functionality, but it cannot guarantee that it is entirely error-free.