Contents

Introduction To Testing

Testing is a detailed investigation process whereby a determination is made whether a piece of code or a product that is intended for customer use is meeting the specifications it's required to.

Software quality may be defined as conformance to explicitly mentioned functional and performance requirements, explicitly documented development standards and implicit characteristics that are expected of all professionally developed software.

Software Quality can be achieved only through continuous improvement which is best illustrated in Deming's cycle also called as PDCA cycle. The plan–do–check–act cycle is a four-step model for carrying out change. Just as a circle has no end, the PDCA cycle should be repeated again and again for continuous improvement.

Verification Vs Validation

Verification is the process of ensuring that software being developed will satisfy the functional specifications and conform to the standards. This process always helps to verify "Are we building the product right?" (According to the functional and technical specifications).

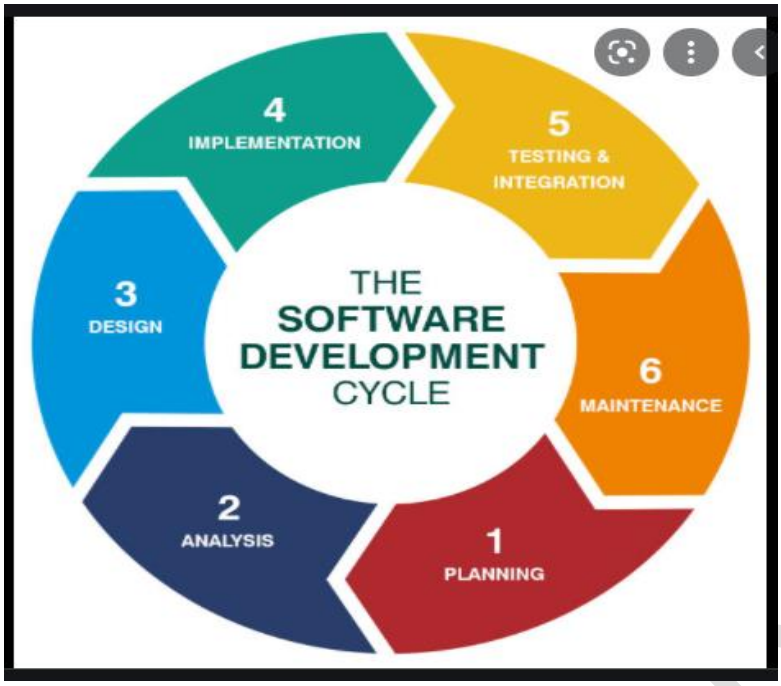
Verification testing ensures that the *expressed* user requirements, gathered in the Project Initiation Phase, have been met in the *Project Execution* phase. One way to do this is to produce a *user requirements matrix* or *checklist* and indicate how you would test for each requirement.

Validation is the process of ensuring that software being developed will satisfy the user needs. This process always helps to verify "Are we building the right product?" (According to the needs of the end user).

Validation testing ensures that any requirement has been met by the end product developed.

SDLC

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.



SDLC Phases:

- **Requirements analysis:** In the Requirements analysis phase, the requirements of the proposed system are collected by analyzing the needs of the user(s). The user requirements document will typically describe the system's functional, physical, interface, performance, data, security requirements etc as expected by the user.
- **Design, Code & Test:** Systems design is the phase where system engineers analyze and understand the business of the proposed system by studying the user requirements document... Technical documentation like entity diagrams, data dictionary, High level design, Low level design, and code for the software will also be produced in this phase. The architecture/design of the software is arrived in during design. The design is converted into software using coding. Each unit is coded and tested individually. The tested units are then integrated and built into a complete product/software.

SDLC Types

Waterfall Model:

The waterfall methodology proceeds from one phase to the next in a sequential manner. For example, one first completes requirements specification, which after sign-off is considered "set in stone." When requirements are completed, one proceeds to design. The software in question is designed and a blueprint is drawn for implementers (coders) to follow—this design should be a plan for implementing the requirements given. When the design is

complete, an implementation of that design is made by coders. Towards the later stages of this implementation phase, separate software components produced are combined to introduce new functionality and reduced risk through the removal of errors.
Thus the waterfall model flows such that moving from one phase to another phase is feasible only if the preceding phase is completed and is perfect.

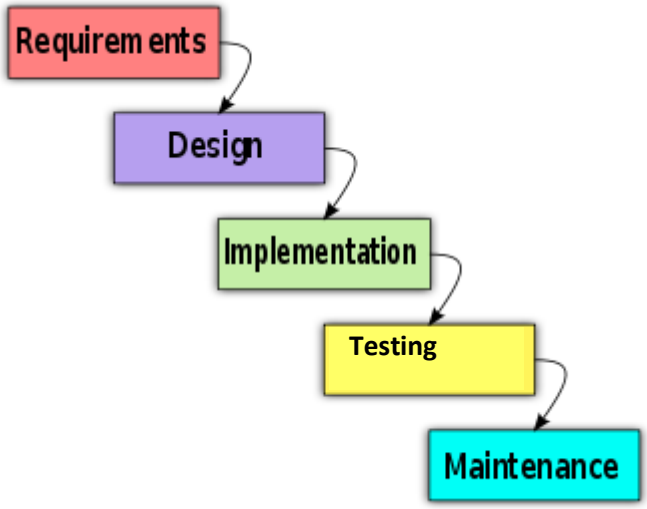


Fig: Waterfall Model

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Iterative Model:

The incremental, or iterative, development/testing model break the project into small parts. Each part is subjected to multiple iterations of the waterfall model. At the end of iteration, a new module is completed or an existing one is improved on, the module is integrated into the structure, and the structure is then tested as a whole. For example, using the iterative development model, a project can be divided into 12 one- to four-week iterations. The system is tested at the end of implementation in each iteration, and the test feedback is immediately incorporated at the end of each test cycle. The time required for successive iterations can be reduced based on the experience gained from past iterations. The system grows by adding new functions during the development portion of iteration. Each cycle tackles a relatively small set of requirements; therefore, testing evolves as the system evolves.

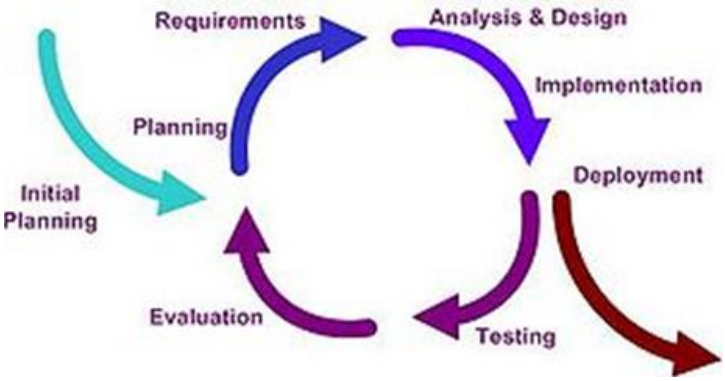


Fig: Iterative Model

Iterative Model - Pros and Cons

The advantage of this model is that there is a working model of the system at a very early stage of development, which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

The advantages of the Iterative and Incremental SDLC Model are as follows −

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.
- Easier to manage risk - High risk part is done first.
- With every increment, operational product is delivered.
- Issues, challenges and risks identified from each increment can be utilized/applied to the next increment.
- Risk analysis is better.
- It supports changing requirements.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.
- During the life cycle, software is produced early which facilitates customer evaluation and feedback.

The disadvantages of the Iterative and Incremental SDLC Model are as follows −

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.

- Management complexity is more.
- End of project may not be known which is a risk.
- Highly skilled resources are required for risk analysis.
- Projects progress is highly dependent upon the risk analysis phase.

Spiral Model:

In the Spiral Model, a cyclical and prototyping view of software development is shown. Test are explicitly mentioned (risk analysis, validation of requirements and of the development) and the test phase is divided into stages. The test activities include module, integration and acceptance tests. However, in this model the testing also follows the coding. The exception to this is that the test plan should be constructed after the design of the system. Also known as the spiral lifecycle model, this model of development combines the features of the prototyping model and the waterfall model. The spiral model is intended for large, expensive and complicated projects.

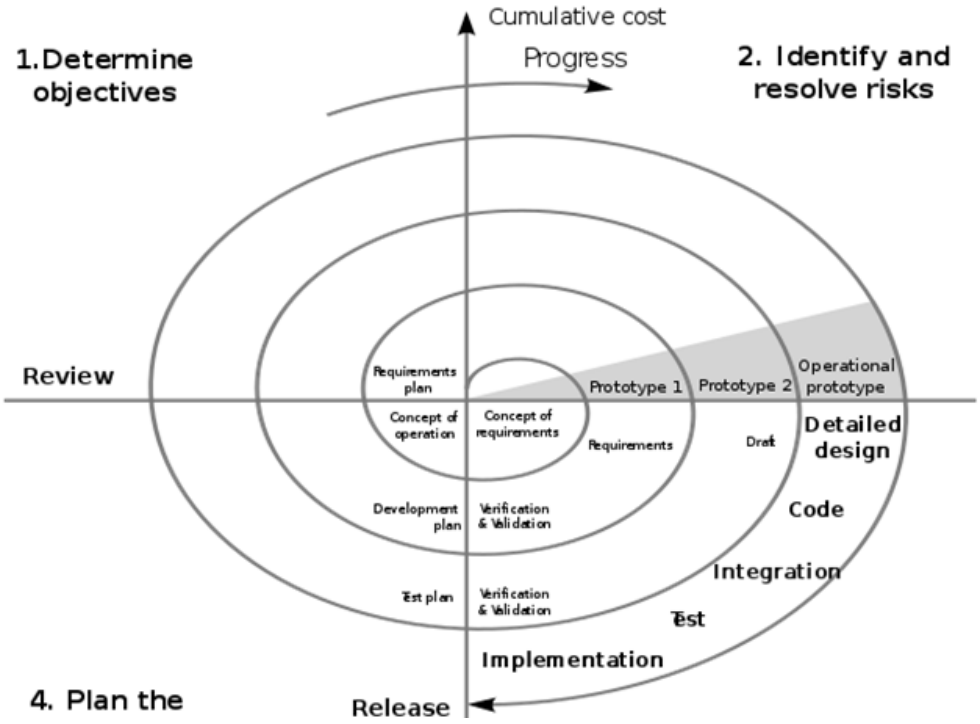


Fig: Spiral Model

Spiral Model - Pros and Cons

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The advantages of the Spiral SDLC Model are as follows −

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows −

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

Agile Methodology:

Agile testing involves testing from the customer perspective as early as possible, testing early and often as code becomes available and stable enough, since working increments of the software are released often in agile software development. Most software development/testing life cycle methodologies are either iterative or follow a sequential model (as the waterfall model does). As software development becomes more complex, these models cannot efficiently adapt to the continuous and numerous changes that occur. Agile methodology was developed to respond to changes quickly and smoothly. Although the iterative methodologies tend to remove the disadvantage of sequential models, they still are based on the traditional waterfall approach. Agile methodology is a collection of values, principles, and practices that incorporates iterative development, test, and feedback into a

new style of development.

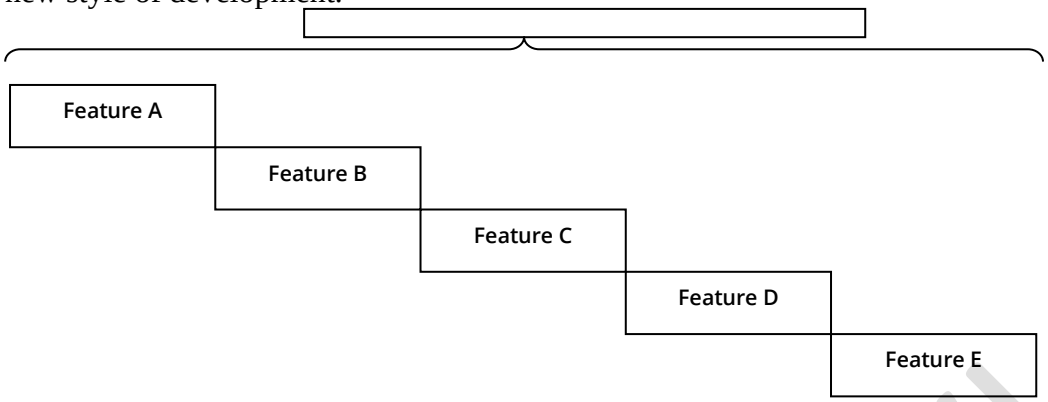


Fig: Agile Model

Agile Model - Pros and Cons

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the Agile model.

The advantages of the Agile Model are as follows −

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

The disadvantages of the Agile Model are as follows −

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.

- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

Verification and Validation model

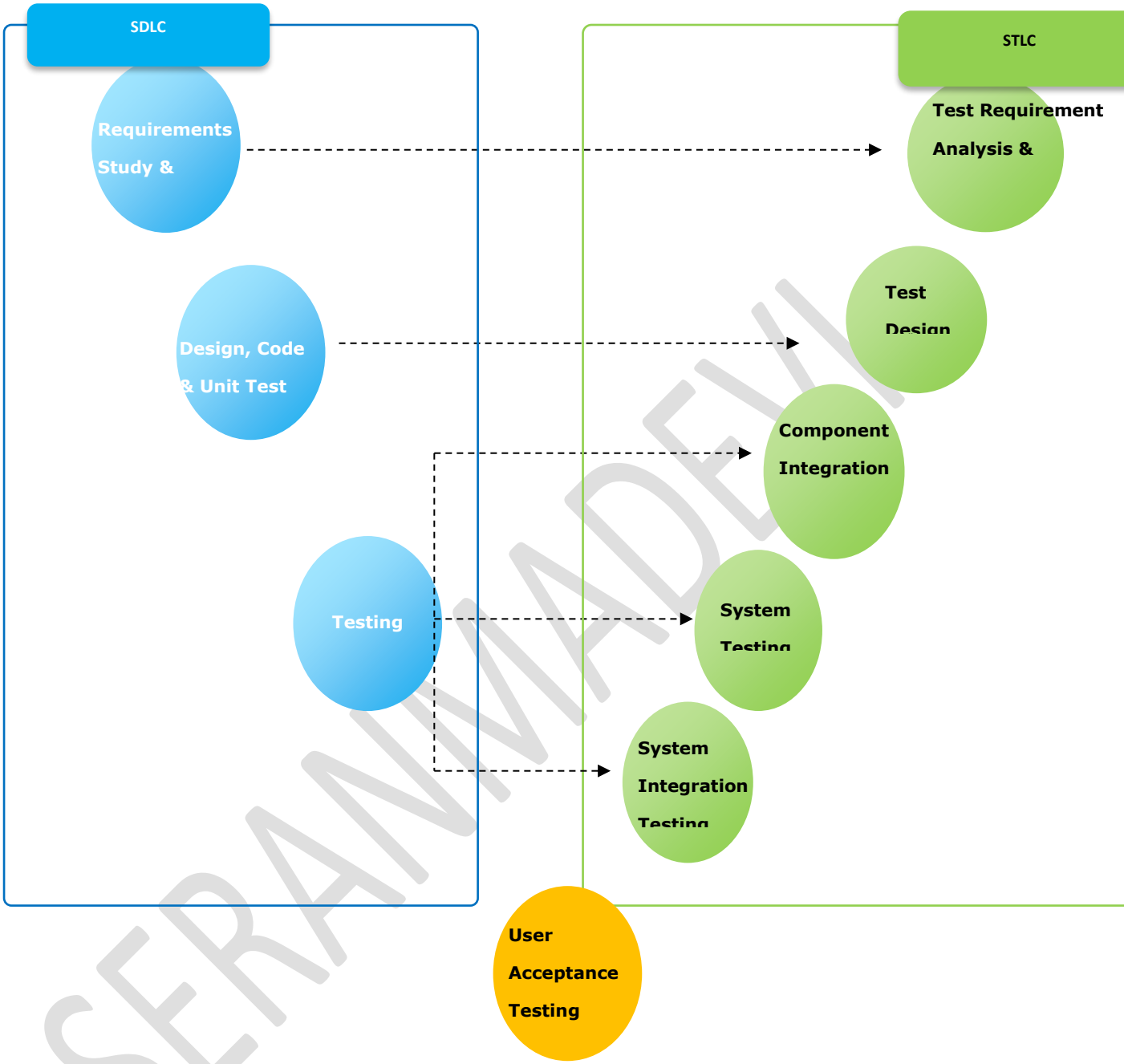
The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**.

The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.

V-Model - Design

Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.

Testing Process aligned to Waterfall development model/ V Model:



V-Model - Pros and Cons

The advantage of the V-Model method is that it is very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today's dynamic world, it becomes very expensive to make the change.

The advantages of the V-Model method are as follows −

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

The disadvantages of the V-Model method are as follows −

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

Software Prototyping

Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Software Prototyping - Pros and Cons

Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final software developed. The model has its own pros and cons discussed as follows.

The advantages of the Prototyping Model are as follows −

- Increased user involvement in the product even before its implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.

- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

The Disadvantages of the Prototyping Model are as follows −

- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much if it is not monitored properly.

General Testing Principles

A number of testing principles have been suggested over the past 40 years and offer general guidelines common for all testing.

Principle 1 – Testing shows presence of Defects:
Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, it is not a proof of correctness. Lesser the number of defects in software, the better is the quality of the software.

Principle 2 – Exhaustive Testing is impossible:
Testing everything (all combinations of inputs and preconditions) is not feasible except for trivial cases. Instead of exhaustive testing, risk analysis and priorities should be used to focus testing efforts if the detailed testing is not feasible for various other reasons.

Principle 3 – Early Testing:
To find defects early, testing activities shall be started as early as possible in the software or system development life cycle, and shall be focused on defined objectives.

Principle 4 – Defect Clustering:
Testing effort shall be focused proportionally to the expected and later observed defect density of modules. A small number of modules usually contains most of the defects discovered during pre-release testing, or is responsible for most of the operational failures.

Principle 5 – Pesticide Paradox:

If the same tests are repeated over and over again, eventually the same set of test cases will no longer find any new defects. To overcome this "pesticide paradox", test cases need to be regularly reviewed and revised, and new and different tests need to be written to exercise different parts of the software or system to find potentially more defects.

Principle 6 – Testing is Context Dependant:
Testing is done differently in different contexts. For example, safety-critical software is tested differently from an e-commerce site.

Principle 7 – Absence-of-errors fallacy:
Finding and fixing defects does not help if the system built is unusable and does not fulfil the users' needs and expectations.

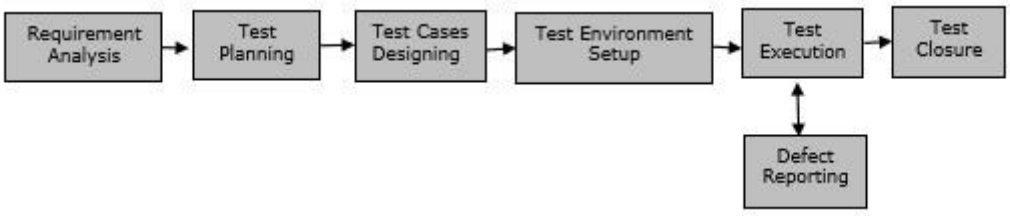
STLC

STLC stands for Software Testing Life Cycle. STLC is a sequence of different activities performed by the testing team to ensure the quality of the software or the product.

- STLC is an integral part of Software Development Life Cycle (SDLC). But, STLC deals only with the testing phases.
- STLC starts as soon as requirements are defined or SRD (Software Requirement Document) is shared by stakeholders.
- STLC provides a step-by-step process to ensure quality software.

STLC Phases

STLC has the following different phases but it is not mandatory to follow all phases. Phases are dependent on the nature of the software or the product, time and resources allocated for the testing and the model of SDLC that is to be followed.



There are 6 major phases of STLC −

- **Requirement Analysis** − When the SRD is ready and shared with the stakeholders, the testing team starts high level analysis concerning the AUT (Application under Test).
- **Test Planning** − Test Team plans the strategy and approach.
- **Test Case Designing** − Develop the test cases based on scope and criteria's.
- **Test Environment Setup** − When integrated environment is ready to validate the product.
- **Test Execution** − Real-time validation of product and finding bugs.
- **Test Closure** − Once testing is completed, matrix, reports, results are documented

Difference between Test Case & Use Case

A **Use Case** is used to define the system that how to use the system for performing a specific task.

A **Test Case** is defined as a group of test inputs, execution condition, and expected results which further lead to developing a particular test objective.

Difference between test case and Test scenario

Test Case	Test Scenarios
The test case is just a document that is detailed which provides details about the assessment method, testing process, preconditions, and anticipated output.	The test Scenarios is just a document that is detailed which provides details about the assessment method, testing process, preconditions, and anticipated output. The test scenarios are the ones based on the use situation and give the one-line information by what to check.
It includes all the positive and inputs being negative navigation measures, anticipated results, pre and post condition, etc.	Test scenarios are one-liner statement, however it is linked to a few test instances.

Difference between Test Plan & Test Strategy

Test Plan	Test Strategy
• A test plan for software project can be defined as a document that defines the scope, objective, approach and emphasis on a software testing effort	• Test strategy is a set of guidelines that explains test design and determines how testing needs to be done
• Components of Test plan include- Test plan id, features to be tested, test techniques, testing tasks, features pass or fail criteria, test deliverables,	• Components of Test strategy includes- objectives and scope, documentation formats, test processes, team reporting

responsibilities, and schedule, etc.	structure, client communication strategy, etc.
• Test plan is carried out by a testing manager or lead that describes how to test, when to test, who will test and what to test	• A test strategy is carried out by the project manager. It says what type of technique to follow and which module to test

How To Write Test Cases in Manual Testing

Follow the below steps to write the test cases.

Step 1 – Test Case ID:

Each test case should be represented by a unique ID. It's good practice to follow some naming convention for better understanding and discrimination purposes.

Step 2 – Test Case Description:

Pick test cases properly from the test scenarios

Example:

Test scenario: Verify the login of Gmail
Test case: Enter a valid username and valid password

Step 3 – Pre-Conditions:

Conditions that need to meet before executing the test case. Mention if any preconditions are available.

Example: Need a valid Gmail account to do login

Step 4 – Test Steps:

To execute test cases, you need to perform some actions. So write proper test steps. Mention all the test steps in detail and in the order how it could be executed from the end-user's perspective.

Example:

- Enter Username
- Enter Password
- Click Login button

Step 5 – Test Data:

You need proper test data to execute the test steps. So gather appropriate test data. The data which could be used an input for the test cases.

Example:

- Username: rajkumar@softwaretestingmaterial.com
- Password: STM

Step 6 – Expected Result:

The result which we expect once the test cases were executed. It might be anything such as Home Page, Relevant screen, Error message, etc.,

Example: Successful login

Step 7 – Post Condition:

Conditions that need to achieve when the test case was successfully executed.

Example: Gmail inbox is shown

Step 8 – Actual Result:

The result which system shows once the test case was executed. Capture the result after the execution. Based on this result and the expected result, we set the status of the test case.

Example: Redirected to Gmail inbox

Step 9 – Status:

Finally set the status as Pass or Fail based on the expected result against the actual result. If the actual and expected results are the same, mention it as Passed. Else make it as Failed. If a test fails, it has to go through the bug life cycle to be fixed.

Example:

Result: Pass

Other important fields of a test case template:

Project Name: Name of the project the test cases belong to

Module Name: Name of the module the test cases belong to

Reference Document: Mention the path of the reference documents (if any such as Requirement Document, Test Plan, Test Scenarios, etc.,)

Created By: Name of the Tester who created the test cases

Date of Creation: When the test cases were created

Reviewed By: Name of the Tester who created the test cases

Date of Review: When the test cases were reviewed

Executed By: Name of the Tester who executed the test case

Date of Execution: When the test case was executed

Comments: Include value information which helps the team

Best Practices To Write Good Test Case

A well-written test case should:

- Easy to understand and execute
- Create Test Cases with End User's perspective
- Use unique test case id
- Have a clear description
- Add proper pre & postconditions
- Specify the exact expected result
- Test cases should be reusable & maintainable
- Utilize testing techniques
- Get peer review

If you follow the best practices to write test cases then anyone in the team can understand and execute the well-written test case easily. It should be easy to read and understand, not only for whoever wrote it but also for other testers as well.

Easy to understand and execute:

To make the test cases easy to understand and execute faster we need to use simple and easy to understand language like "Go to login page", "enter username", "enter password", "click on login button" and so on.

Create Test Cases with End User's perspective:

Create test cases by keeping end-user in mind. The test cases you create must meet customer requirements.

Use unique Test Case ID:

It's good practice to follow a unique id with some naming convention for better understanding and discrimination purposes.

Have a clear description:

Your test case description should be clear enough to understand what the tester is going to do with this test case.

Add proper preconditions & postconditions:

In some cases, test cases need to meet some conditions before execution or achieve some conditions after execution. These conditions we need to mention properly in the Pre and postconditions.

Specify the exact expected result:

Include the Expected result: Expected result tells us what will be the result of a particular test step. Testers decide the pass or fail criteria based on the expected result.

Test cases should be reusable & maintainable:

A well written test case is reusable and maintainable. There ar times where developers change the code, and testers need to update the test cases. If our test cases are easy to read and understand then it would be easy to update them not only by whoever wrote it but also by other testers as well.

Requirement Traceability Matrix (RTM)

RTM is a document that maps and traces user requirement with test cases. It captures all requirements proposed by the client and requirement traceability in a single document, delivered at the conclusion of the Software developement life cycle. The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

Test Case Design Technique

The various Test Design Techniques that can be used for creating test cases are as follows:

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table
- State Transition Diagram

Equivalence Partitioning

Equivalence Partitioning is a software testing technique that divides the input data of a software unit into partitions of data from which test cases can be derived.

Equivalence class is a subset of data that is representative of a larger class

It divides the input data of a software unit into partitions of data from which test cases can be derived

It attempts to cover classes of errors

In principle, test cases are designed to cover each partition at least once. So Test cases should be chosen from each partition

Example

In a website, the Total Amount for paying the monthly credits accepts credit limits with a given range say: $10,000 – $15,000

This would have three equivalence classes:

1. Less than $10,000 (Invalid)
2. Between $10,000 and $15,000 (Valid)
3. Greater than $15,000 (Invalid)

The Test Cases shall be chosen from the above equivalence classes

Boundary Value Analysis

Boundary Value Analysis is a technique that consists of developing test cases and data that focuses on the input and output boundaries of a given function.

Complements equivalence partitioning and selects the test cases at the “edge” of equivalence classes

Identify errors at boundaries rather than finding those that exist in center of input domain

Tests are designed to include representatives of boundary values

Example

In the same credit limit example stated earlier, the Boundary Value Analysis would test the following:

1. Low boundary plus or minus one ($ 9,999 and $10,001)
2. On the boundary ($10,000 and $15000)
3. Upper boundary plus or minus one ($14999 and $15001)

Following are the range of Boundary Values:

Value immediately below range

First value of range

Second value of range

Value immediately below last value of range

Last value of range

Value immediately above range

Decision Table

A Decision table lists causes and effects in a matrix. Each column represents a unique combination.

The decision table allows the iteration of all the combinations of values of the condition, thus it provides a “completeness check.”

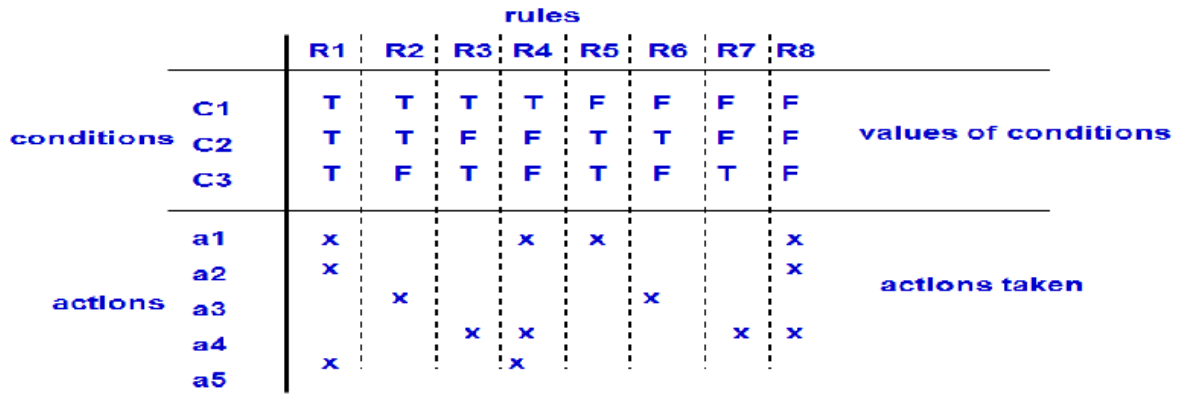
The conditions in the decision table may be interpreted as the inputs, and the actions may be thought of as outputs.

Purpose is to structure logic

Precise yet compact way to model complicated logic

Cause = condition

Effect = action = expected results



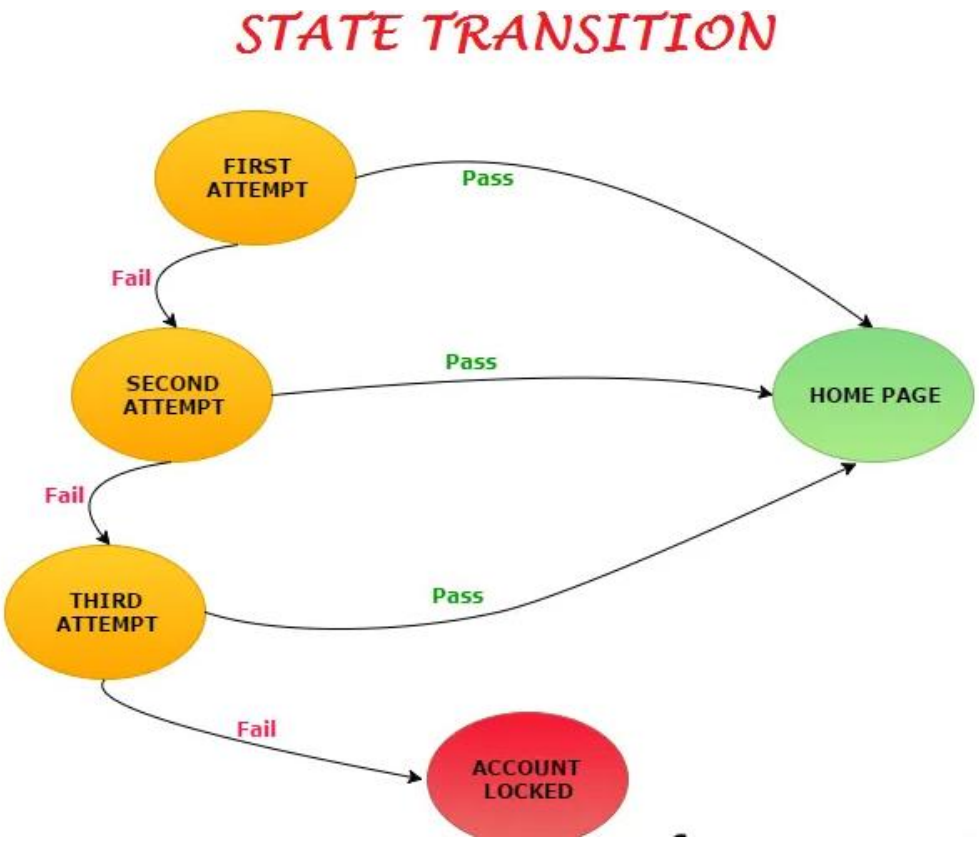
State Transition

Using state transition testing, we pick test cases from an application where we need to test different system transitions. We can apply this when an application gives a different output for the same input, depending on what has happened in the earlier state.

Example on State Transition Test Case Design Technique:

Take an example of login page of an application which locks the user name after three wrong attempts of password.

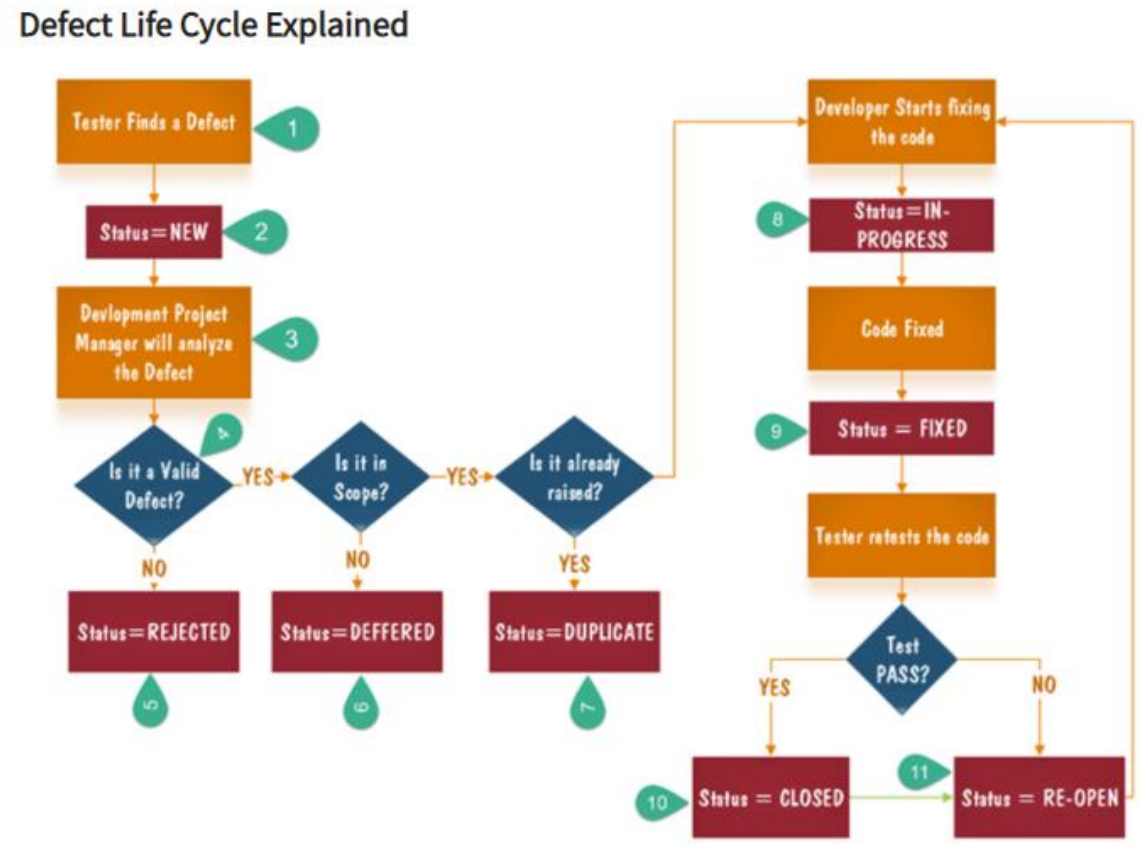
A finite state system is often shown as a state diagram



It works like a truth table. First determine the states, input data and output data

STATE	LOGIN	CORRENT PASSWORD	INCORRECT PASSWORD
S1	First Attempt	S4	S2
S2	Second Attempt	S4	S3
S3	Third Attempt	S4	S5
S4	Home Page		
S5	Display a message as "Account Locked, please consult Administrator"		

Defect Life Cycle



Bug Reporting Template

- **ID/name**: Keep it brief and use correct terms. A best practice is to include the name of the feature where you found an issue. A good example could be 'CART - Unable to add new item to my cart'.
- **Description/summary:** If you feel the name is not sufficient, explain the bug in a few words. Share it in easy-to-understand language. Keep in mind that your description might be used to search in your bug tracking application, so make sure to use the right words.
- **Environment:** Depending on your browser, operating system, zoom level and screen size, websites may behave differently from one environment to another. Make sure your developers know your technical environment.
- **Console logs:** By collecting the console logs your developers will find it a lot easier to reproduce and resolve any bug.
- **Source URL:** Make it easy for your developers spot the problem by including the URL of the page where you found the bug. Big time saver!
- **Visual proof:** A picture is worth a thousand words. Although it might not be enough, a visual element like a screenshot or a video will help your developers understand the problem better and faster.

- **Steps to reproduce:** A screenshot is a proof that you had a problem, but keep in mind that your developer might not be able to reproduce the bug. Make sure to describe, with as much detail as possible, the steps you took before you encountered the bug.
- **Expected vs. actual results:** Explain what results you expected - be as specific as possible. Just saying "the app doesn't work as expected" is not useful. It's also helpful to describe what you actually experienced.

Optional: You can also include extra information such as the severity (critical, major, minor, trivial, enhancement), priority (high, medium, low), name of the reporter, person assigned or a due date.

Bugs can be reported in a number of ways. However, using a bug tracker is probably the best way for your organization to move bugs from reported to fixed and help your developers stay focused.

Difference between Defect, Error, Bug, Failure

Testing is the process of identifying defects, where a defect is any variance between actual and expected results. "A mistake in coding is called Error, error found by tester is called Defect, defect accepted by development team then it is called Bug, build does not meet the requirements then it Is Failure."

Severity – How much the application got impacted because of that defect - testers

Priority- How soon the defect needs to be fixed - BA

Showstopper- we cannot progress because of a defect Eg: URL launch

Critical – we can progress up to a certain level

Major

Minor- sporadic defect – spelling mistake – It won't affect the flow