Learning Outcome

## Software Development Life Cycle (SDLC)

# SDLC Overview

## What is SDLC?

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

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.

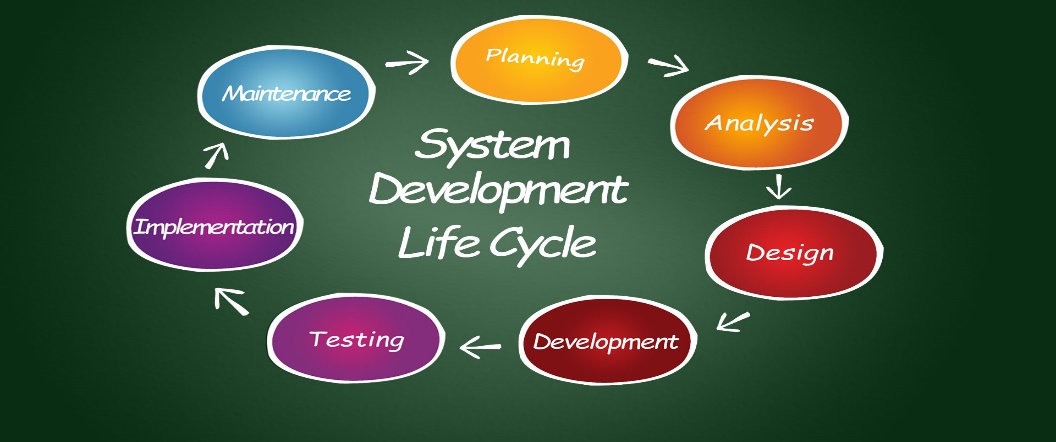


Image 1: SDLC

Reference: <https://svitla.com/blog/system-development-life-cycle>

### **Why SDLC?**

* It offers a basis for project planning, scheduling, and estimating
* Provides a framework for a standard set of activities and deliverables
* It is a mechanism for project tracking and control
* Increases visibility of project planning to all involved stakeholders of the development process
* Increased and enhance development speed
* Improved client relations
* Helps you to decrease project risk and project management plan overhead

## Phases of Software Development Life Cycle (SDLC)

The entire SDLC process divided into the following SDLC steps:

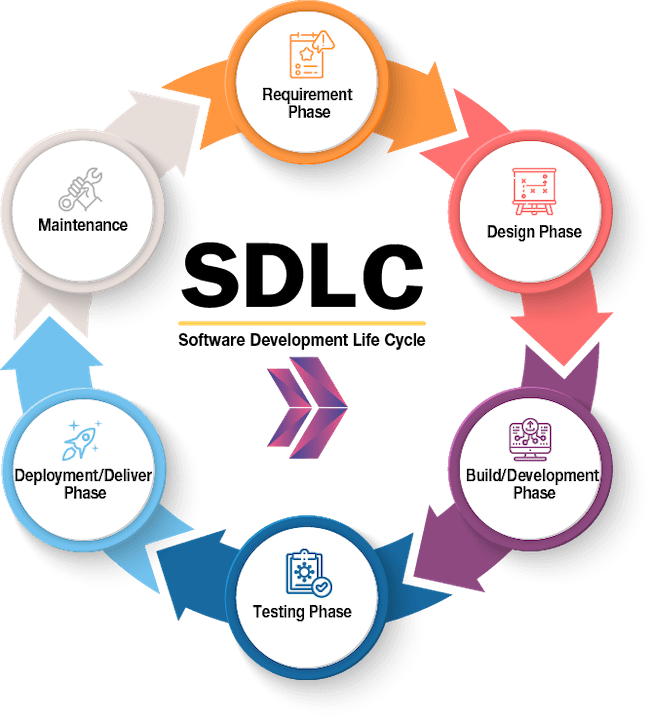


Image 2: SDLC Life Cycle

Reference: <https://www.slideshare.net/webdevninja/introduction-to-css-13603389>

* Phase 1: Requirement collection and analysis
* Phase 2: Feasibility study
* Phase 3: Design
* Phase 4: Coding
* Phase 5: Testing
* Phase 6: Installation/Deployment
* Phase 7: Maintenance

**Phase 1: Requirement collection and analysis**

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

**Phase 2: Feasibility study**

Once the requirement analysis phase is completed the next sdlc step is to define and document software needs. This process conducted with the help of ‘Software Requirement Specification’ document also known as ‘SRS’ document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

* Economic: Can we complete the project within the budget or not?
* Legal: Can we handle this project as cyber law and other regulatory framework/compliances.
* Operation feasibility: Can we create operations which is expected by the client?
* Technical: Need to check whether the current computer system can support the software
* Schedule: Decide that the project can be completed within the given schedule or not.

**Phase 3: Design**

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

**High-Level Design (HLD)**

* Brief description and name of each module
* An outline about the functionality of every module
* Interface relationship and dependencies between modules
* Database tables identified along with their key elements
* Complete architecture diagrams along with technology details

**Low-Level Design (LLD)**

* Functional logic of the modules
* Database tables, which include type and size
* Complete detail of the interface
* Addresses all types of dependency issues
* Listing of error messages
* Complete input and outputs for every module

**Phase 4: Coding**

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

**Phase 5: Testing**

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

**Phase 6: Installation/Deployment**

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

**Phase 7: Maintenance**

Once the system is deployed, and customers start using the developed system, following 3 activities occur

* Bug fixing – bugs are reported because of some scenarios which are not tested at all
* Upgrade – Upgrading the application to the newer versions of the Software
* Enhancement – Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

## Software Development Life Cycle Models

Software development (SDLC) can be performed according to a variety of models, varying based on project type, requirements given, time constraints, resources, and broader company product development objectives.

We introduce six of the most popular models, including

* waterfall,
* iterative,
* V-Shaped
* Spiral
* Agile
* Big Bang.

**SDLC Model: Waterfall**

The waterfall model is seen as one phase “flowing” into the next as any phase in the process cannot commence until the previous is completed. The most commonly accepted of models is documentation-intensive as the earlier documentation tells what will be performed in future phases. Possibly the most straightforward of models, it is easy to manage, although flaws cannot be corrected until the “maintenance” stage.

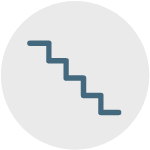


Image 3: Waterfall model

Reference: <https://sdlcpartners.com/insights/what-are-sdlc-types/>

**Uses:**

Waterfall is typically made up of seven consecutive phases: requirement gathering, feasibility analysis, design, coding, testing, installation, and maintenance. Often used for simple, small or mid-sized projects, waterfall has traditionally been seen as offering a stricter control mechanism. It can be appropriate for smaller projects where the technology is understood and the requirements are well defined, fixed.

**Pros:**

* Simple to use and understand
* Easy to classify and prioritize tasks
* Well understood milestones and checkpoints
* Each phase has specific deliverables

**Cons:**

* High risks and uncertainty
* Assumes the requirements of a system can be frozen
* Difficult to go back to any stage once it is finished
* Difficult to measure progress within stages

**SDLC Model: Iterative**

The iterative model is a realization of the sequential approximation method, meaning it is a series of waterfall models where the requirements are divided into groups at the beginning of the project. The main difference from the previous is that the SDLC can start without all of the requirements gathered. Developers implement a set of software requirements, test, evaluate and pinpoint further requirements. This model provides a working version earlier in the cycle.

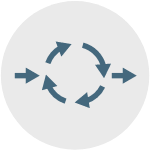
****

Image 4: Iterative model

Reference: <https://sdlcpartners.com/insights/what-are-sdlc-types/>

**Uses:**

Applied to large-scale projects where the final product and main task are predefined, but the details evolve over time. This model is well-suited for projects that have a time constraint as well as large, mission-critical projects that consist of loosely coupled parts like microservices or web services.

**Pros:**

* Risk analysis is more thorough
* Initial operating time is faster
* More focused on customer value than linear approaches
* Encourages flexibility and readiness to change to evolving requirements

**Cons:**

* More resources may be needed
* Complicated to manage
* End of project may not be known, which is a risk
* Partitioning the functions and features may be problematic

**SDLC Model: V-Shaped**

The V-shaped model addresses validation and verification through parallel processes where coding is connected to both concurrently. This model grew out of the waterfall model where each stage begins only after the previous one has ended.

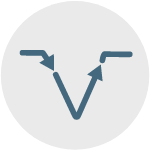


Image 5: V-shaped model

Reference: <https://sdlcpartners.com/insights/what-are-sdlc-types/>

**Uses:**

Most appropriate when failures and downtimes are unacceptable (e.g., medical software, aviation fleet management software) or where there are no unknown requirements because this model is restrictive to making changes mid-cycle.

**Pros:**

* Testing and verification takes place in early stages
* Easy to control
* Highly disciplined
* Good when requirements are static and clear

**Cons:**

* Lack of flexibility
* Meant only for bigger projects
* Not suitable for projects where requirements are likely to change
* Once in the testing phase, it is difficult to go back and change aspects like functionality

**SDLC Model: Spiral**

Iterative in nature, the spiral is a combination of prototype and waterfall models. This is a risk-driven model that is more flexible and allows for building a highly customized product where user feedback can be incorporated early in the project.



Image 6: Spiral model

Reference: <https://sdlcpartners.com/insights/what-are-sdlc-types/>

**Uses:**

In the spiral model, the project passes through four phases over and over in a “spiral” until complete, allowing for multiple rounds of refinement. It is best used in large projects and systems that contain small phases or identifiable segments.

**Pros:**

* Development process is well-documented and scalable
* Progress is easily measured
* High-risk tasks are completed first
* Early involvement of developers

**Cons:**

* Can be expensive, creating a high cost and longer time to reach a final product
* Can be ineffective for smaller projects
* Highly customized, which limits reusability
* Needs special skills to evaluate the risks and assumptions

**SDLC Model: Agile**

Agile has gained a great deal of popularity. The model breaks down big projects into smaller, more manageable chunks, which can lead to a software product that represents a culmination of multiple, smaller projects. The model produces ongoing releases where each iteration includes small, incremental changes and improvements from the previous release.

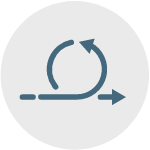


Image 7: Agile model

Reference: <https://sdlcpartners.com/insights/what-are-sdlc-types/>

**Uses:**

Agile is particularly well-suited for large and complex projects; those that can be easily divided into smaller parts. It’s also useful for mid-size custom software development projects where business requirements cannot be easily translated into detailed requirements.

**Pros:**

* Face-to-face communication and continuous inputs from customer representative leave no space for guesswork
* End result is the highest quality software in the least possible time
* Very realistic approach to development
* Suitable for fixed and evolving requirements

**Cons:**

* Documentation is created in later stages
* Reduce the usability of components
* Requires special skills and ongoing oversight
* Transfer of technology to new team members may be difficult due to lack of documentation

**SDLC Model: Big Bang**

Done with little-to-no planning, the Big Bang model focuses on all types of coding and development types, implementing requirements as they are discovered. Because it does not follow a set process and is a high-risk model, the Big Bang is best for small projects with only one or two engineers.

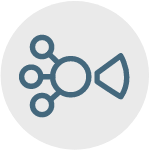


Image 8: Big Bang model

Reference: <https://sdlcpartners.com/insights/what-are-sdlc-types/>

**Uses:**

It is mainly used for academic software development projects or smaller projects where the development team is small and working in tight collaboration. It is helpful when requirements are unknown and a release date is very flexible.

**Pros:**

* Allows for more resources to be used in development
* More straightforward to manage
* Allows developer flexibility
* Good learning aid for students or newcomers

**Cons:**

* Harder to attain a firm grasp on requirements
* High-risk with high level of uncertainty
* Poor model type for long or ongoing projects
* Can be expensive if requirements are misunderstood and timelines are vague

## Software Test Levels

### 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.

### Types of Software Testing

**Manual Testing:**

Manual testing is the process of testing the software by hand to learn more about it, to find what is and isn’t working.

This usually includes verifying all the features specified in requirements documents, but often also includes the testers trying the software with the perspective of their end user’s in mind.

Manual test plans vary from fully scripted test cases, giving testers detailed steps and expected results, through to high-level guides that steer exploratory testing sessions.

There are lots of sophisticated tools on the market to help with manual testing, but if you want a simple and flexible place to start, take a look at Testpad.

**Automation Testing:**

Automation testing is the process of testing the software using an automation tool to find the defects.

In this process, testers execute the test scripts and generate the test results automatically by using automation tools.

Some of the famous automation testing tools for functional testing include Selenium and Katalon Studio.

Selenium is no longer a strange name for web application testers. It offers powerful capabilities like cross-browser testing but is difficult to learn for those new to automation or with limited programming experience. That’s why most QAs freshers and manual testers start out with Katalon Studio. While still providing essential features from Selenium, users can utilize its simple UI, built-in keywords, and record & playback to create test cases easier and pick up programming skills along the way with the scripting mode (Java and Groovy supported).

### Levels of Testing

There are mainly four Levels of Testing in software testing:

* **Unit Testing**: checks if software components are fulfilling functionalities or not.
* **Integration Testing**: checks the data flow from one module to other modules.
* **System Testing**: evaluates both functional and non-functional needs for the testing.
* **Acceptance Testing**: checks the requirements of a specification or contract are met as per its delivery.

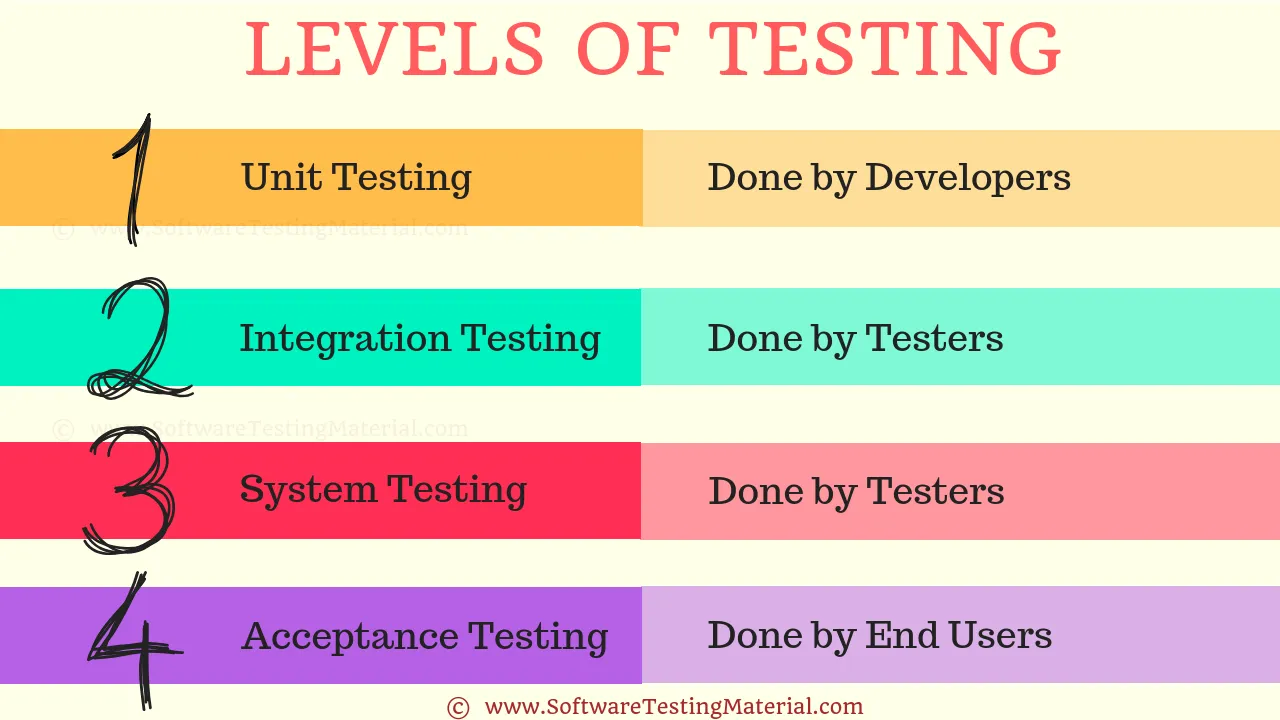


Image 9: Levels of Testing

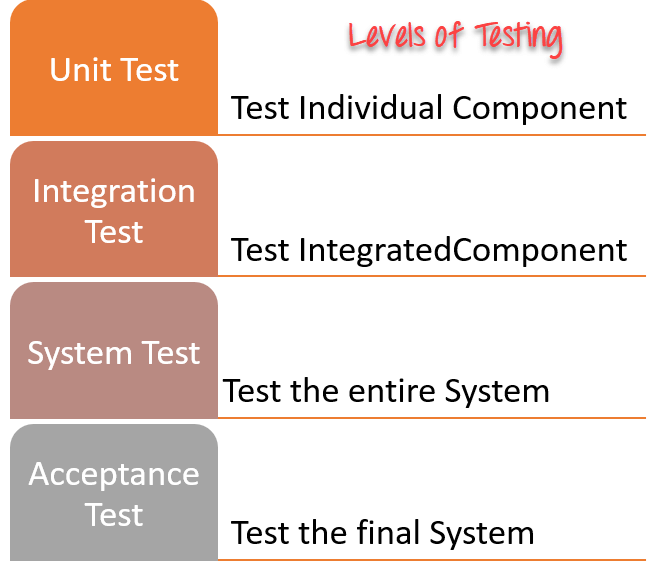


Image 10: Levels of Testing

Reference: <https://www.guru99.com/levels-of-testing.html>

**Unit Testing:**

Unit Testing is done to check whether the individual modules of the source code are working properly. i.e. testing each and every unit of the application separately by the developer in the developer’s environment. It is AKA Module Testing or Component Testing

**Integration Testing:**

Integration Testing is the process of testing the connectivity or data transfer between a couple of unit tested modules. It is AKA I&T Testing or String Testing

It is subdivided into the Top-Down Approach, Bottom-Up Approach and Sandwich Approach (Combination of Top Down and Bottom Up). This process is carried out by using dummy programs called Stubs and Drivers. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

* **Big Bang Integration Testing:**

In Big Bang Integration Testing, the individual modules are not integrated until all the modules are ready. Then they will run to check whether it is performing well. In this type of testing, some disadvantages might occur like, defects can be found at the later stage. It would be difficult to find out whether the defect arouses in an interface or in a module.

* **Top-Down Integration Testing**

In Top-Down Integration Testing, the high-level modules are integrated and tested first. i.e Testing from the main module to the submodule. In this type of testing, Stubs are used as a temporary module if a module is not ready for integration testing.

* **Bottom-Up Integration Testing**

In Bottom-Up Integration Testing, the low-level modules are integrated and tested first i.e. Testing from sub-module to the main module. Same like Stubs, here drivers are used as a temporary module for integration testing.

**Stub:**

It is called by the Module under Test.

**Driver:**

It calls the Module to be tested.

**System Testing (End TO End Testing):**

It’s a black box testing. Testing the fully integrated application this is also called as an end-to-end scenario testing. To ensure that the software works in all intended target systems. Verify thorough testing of every input in the application to check for desired outputs. Testing of the users’ experiences with the application.

**Acceptance Testing:**

To obtain customer sign-off so that software can be delivered and payments received.

Types of Acceptance Testing are

* Alpha Testing
* Beta Testing
* Gamma Testing.

**Alpha Testing**:

Alpha testing is mostly like performing usability testing which is done by the in-house developers who developed the software. Sometimes this alpha testing is done by the client or outsiders with the presence of developers or testers.

**Beta Testing:**

Beta testing is done by a limited number of end users before delivery, the change request would be fixed if the user gives feedback or reports defect.

**Gamma Testing:**

Gamma testing is done when the software is ready for release with specified requirements; this testing is done directly by skipping all the in-house testing activities.

References

* <https://www.guru99.com/software-development-life-cycle-tutorial.html>
* <https://www.guru99.com/levels-of-testing.html>