**Software Development Life Cycle**

Table of Contents

[Requirement Gathering and Analysis 3](#_Toc83392872)

[How Requirements are Gathered 3](#_Toc83392873)

[Challenges 5](#_Toc83392874)

[Feasibility 6](#_Toc83392875)

[Design 7](#_Toc83392876)

[Importance of Software Design 8](#_Toc83392877)

[Coding 9](#_Toc83392878)

[Testing 10](#_Toc83392879)

[Installation and Deployment 11](#_Toc83392880)

[Maintenance 12](#_Toc83392881)

The Software Development Life Cycle (SDLC) is a **systematic process** for building software that ensures **quality** and **correctness** of the software built. SDLC aims to produce **high-quality** software that **meets customer expectations**.

SDLC is used because:

* It offers a basis for project planning, scheduling and estimating.
* It provides a framework for a standard set of activities and deliverables.
* It is a mechanism for project tracking and control.
* It increases visibility of project planning to all involved stakeholders (anyone positively or negatively affected by the project).
* It increases and enhances development speed.
* It improves client relations.
* It helps decrease project risk and project management plan overhead.

The phases of SDLC are:

1. Requirement Gathering and Analysis
2. Feasibility
3. Design
4. Coding
5. Testing
6. Installation and Deployment
7. Maintenance

## Requirement Gathering and Analysis

The Requirement Gathering and Analysis phase is the basis of the whole software development process. In the requirement gathering phase, all business requirements are gathered from the client. In the analysis phase, the requirements are analysed to evaluate whether to proceed with the project. Software specifications are prepared in the **Software Requirement Specification (SRS)** report. If the requirements gathered are incorrect, the project may very well fail.

The requirement gathering phase is important since it allows all **project requisites** to be found so that a full-fledged product with all the specifications can be made. It also allows project associates to **communicate effortlessly** with each other over defined client needs.

### How Requirements are Gathered

Requirements are gathered by **Business Analysts**, who are professionals who can efficiently carry out software requirement gathering by breaking down critical technical specifications into **effective documentation** and **user stories**.

Some tools and techniques that can be used to gather requirements are:

* **Interviews** – The business analyst speaks with the users and clients of the system to be built to figure out what they need from the system.
* **Survey** – A questionnaire is sent to users and stakeholders to collect information.
* **Brainstorming** – Experts on the subject conduct brainstorming sessions to discuss solutions.
* **Joint Application Method** – All stakeholders, such as developers, users, business analysts and software engineers, attend a workshop to discuss details of the system in depth.
* **Observation** – Someone from the development team, the observer, observes the users in their working environment to develop ideas about the system to be built and subsequently documents the observations.
* **Focus Group** – Ideas are collected from representatives of clients and users to understand the software clearly.
* **Interface Analysis** – This is a special technique where specific requirements related to application development are gathered and their interaction with other software components is measured. For example, to build a banking application, existing banking applications could be observed to gather ideas.
* **Prototyping** – A model of the software is built, which helps uncover and capture software requirements from the client.
* **Use Case Diagram** – This is a formal diagram which shows how people interact with the system. It shows what the system does.

### Challenges

Some common challenges faced when gathering requirements are:

* **Lack of clarity** in defining criteria for success.
* Clients **change their minds** frequently.
* **Lack of or over-communication by clients**. In some cases, the client disappears after the initial interaction while in other cases, they constantly keep asking for updates.
* Clients get **stuck on certain techniques** or solutions, perhaps insisting that a specific method be used, even if the same goal can be achieved easily with other techniques.
* Stakeholders can have **conflicting priorities**.

## Feasibility

There are five types of feasibility that we need to check for:

1. **Economic** – This is just the budget.
2. **Legal** – Ensuring all cyber laws and other regulatory frameworks are abided by.
3. **Operational** – Ensuring the operations expected by the client can be delivered.
4. **Technical** – Ensuring the technical skills and software or hardware required to build the project is available.
5. **Schedule** – This refers to time constraints.

## Design

Design documents are prepared based on the SRS. They help define the overall system architecture.

Design documents can be of two types, **High-Level Design (HLD)** and **Low-Level Design (LLD)**, also called **Detail-Level Design (DLD)**.

An HLD:

* Gives a **brief description** of and **names** each module
* Outlines the **functionality** of each module, but does not go into the details of how it works
* Highlights the **relationships** and **dependencies** between modules.
* Identifies **database tables**, along with their **key elements**.
* Shows **complete architecture diagrams**, along with **technology details**.

An LLD:

* Describes the **functional logic** of each module
* Includes **database tables**, along with their type and size
* Shows the complete details of the **interfaces**
* Addresses all **dependency issues**
* Lists **error messages**
* Shows the complete **inputs** and **outputs** of every module

### Importance of Software Design

* **Modularity** – Modular software makes it easy to move or remove parts if needed, or even share the design work between multiple developers.
* **Performance** – Design should describe how the system works internally and how it uses resources like threads, database connections, etc. that might affect the performance.
* **Usability** – Design is a good starting point. Instead of going through all the source code, the software design can give all the information required to get started.
* **Maintainability** – Good software design makes it easier to maintain software. It can immediately be seen how much change would be needed to fix a bug or add a new feature.
* **Portability** – Including dependencies or other software modules, such as third-party libraries, makes it easier to port software into another environment.
* **Trackability** – Good design tracks the requirements and proves on the design level that all requirements are being met.

## Coding

In the coding stage, developers start to **build the system** using the **chosen programming language**. Tasks are **divided into units**, or modules, and **assigned** to various developers. The Coding stage is the **longest phase** of the SDLC.

Developers need to follow certain **predefined coding guidelines** and use **programming tools** like compilers, interpreters and debuggers to generate and implement code.

There are a few things that are essential to follow in the Coding phase:

1. Using some **version control system**.
2. Before coding, spending some time to select a **development tool** that will be suitable for debugging, coding and modification.
3. Before coding, defining some **standard**, since multiple developers will be using the code.
4. During development, writing appropriate **comments** so that others understand the logic behind the code.
5. Having regular **review meetings** to identify defects early on.

## Testing

In the Testing phase, we focus on **investigation** and **discovery**. We first define **how** the software will be tested and **who** will test it. While testing, we try to find whether the developed code works according to **customer requirements**.

The QA and testing teams may find some **bugs** or defects that they communicate with the developers. The developers must then **fix** the bugs and send the software back to the testing teams for a **re-test**. This process continues until the software is **bug-free**, **stable** and working according to the **business needs** of the system.

## Installation and Deployment

Deployment is the process of promoting a **fully-tested** and **approved** application. The QA manager has finalized the final **bug-free release** at this stage. The deployment guide is provided to the **operations team** for hassle-free deployment.

Deployment could require different promotion processes:

* For **web applications**, the application would need a **server**, which would also require testing and redirecting the DNS.
* For **standalone applications**, media or **downloadable packages** would need to be produced.
* For **software as a service**, **licensing issues** would need to be determined.

Finally, deployment also requires that **data migration** to the live server has occurred successfully.

## Maintenance

The Maintenance stage covers all activities that take place after the software has been deployed and is in use by end-users.

Major maintenance activities include:

* **Bug Fixing** – Bugs reported by users in scenarios that were not tested must be fixed.
* **Upgrade** – The application must be periodically updated to newer versions, for example for security purposes.
* **Enhancement** – New features could be added to the existing software.

Essentially, we must ensure that the **needs** of the users **continue to be met** and the system **continues to perform** as per the specifications mentioned in the first phase.