# Introduction

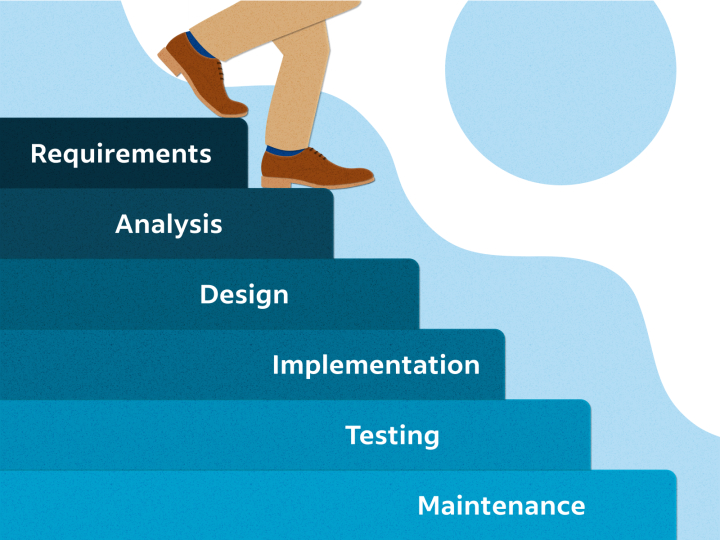
A general Software Development Life Cycle (SDLC) looks like this:

SDLC is just a blueprint to develop a software product. In order to apply it to specific projects, we need models that apply SDCL phases. They will be detailed guidelines for team colloboration.

# Waterfall Model

## What Is It?

The Waterfall model is a Software Development Process which appears most intuitive to us at the first sight. You gather requirements for a project. Once the requirements are complete, you complete the design. When the design in complete, you write code and complete development. And so on …



All planning is done first with detailed documentation and the scope of work is generally fixed (because you’ll freeze the scope of one phase before starting the next).

## What Are Pros and Cons?

### Pros

### Cons

* The output of a phase becomes input of the next phase. **If you introduce an error in one phase, that error will propagate to all other phases**. As a result, to fix that, you have to go back to the early phases and start **re-working**.
* The **customer does not get to see the product before the early testing phase** which is 2/3 the way through the product time line.
* When you reach to the Deployment and Maintenance phase, you could realize that **your product was no longer viable due to change in market conditions**, etc.
* You could realize that the **product had a major architectural flaw that prevented it from being deployed**. In other words, your product development could completely fail after lots of money and time.

In software development, everything changes: requirements, skills, people, environment, etc. **Software development is inherently an iterative process and does not work like a Waterfall cycle**.

## How People Tried to Fix It?

When Waterfall projects started failing, many teams treated this failure as if there was a failure in a production factory. So they tried to fix it by:

* Add more **comprehensive documentation**. Having a well-documented software system is good. But the documentation by itself adds no value to the stakeholders.
* Maintain **comprehensive checklists** to make sure they were producing systems of high quality. Checklists, like coding standards and architectural reviews, are helpful. But you cannot produce a single recipe book for building software.

However, they forgot that more time should be spent on maneuvering working software features early and often, as well as enlisting customer feedbacks.

## When To Use It?

Waterfall model is highly applicable when:

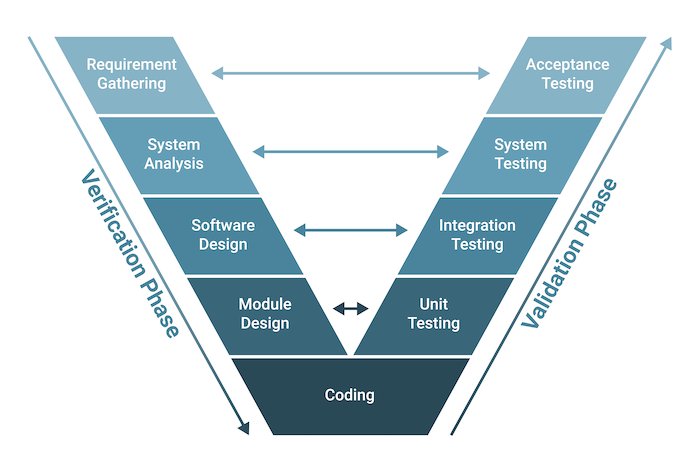
* The software system is **simple and small**.
* The software system reached **maintenance phase**. This is specifically applicable if the development team has good domain knowledge and both business and stakeholders are good at working with each other.
* The software system is a **mission critical system** (gated phases required to avoid catastrophic failures). An example is a system where a defect can cause human causality. Comprehensive documentation is also very applicable here.

# V Model

## What Is It?

The V model is a Software Development Process where development phases are executed in a sequential manner in a V-shape. It’s also known as Verification and Validation model. And it’s considered as a **sub-set of Waterfall 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.



### Verification Phase

The verification phase refers to the practice of evaluating the product development process to ensure the team meets the specified requirements.

It includes several steps:

* **Requirement Gathering (or Business Requirement Analysis)**: The team comes to understand the product requirements by the customer.
* **System Analysis**: The system engineers analyze and interpret the business requirements of the proposed system by studying the user requirements document.
* **Software Design**: The team selects the software architecture based on the list of modules, brief functionality of each module, interface relationships, dependencies, database tables, architecture diagrams, technology detail and more. The integration testing model is developed in this phase.
* **Module Design**: The development team breaks down the system into small modules and specifies the detailed design of each module, which we call low-level design.
* **Coding**: The development team selects a suitable programming language based on the design and product requirements. There are, of course, guidelines and standards for coding and the code will go through many reviews to check its performance.

### Validation Phases

The validation phase involves dynamic analysis methods and testing to ensure the software product meets the customer’s requirements and expectations.

It includes several stages:

* **Unit Testing**: The team develops and executes unit test plans to identify errors at the code or unit level. This testing happens on the smallest entities, such as program modules, to ensure they function correctly when isolated from the rest of the code.
* **Integration Testing**: It involves executing integration test plans developed during the architectural design step in order to verify that groups created and tested independently can coexist and communicate with each other.
* **System Testing**: It involves executing system test plans developed during the system design step, which are composed by the client’s business team. System testing ensures the team meets the application developer’s expectations.
* **Acceptance** **Testing**: It’s related to the business requirement analysis part of the model and involves testing the software in the user environment to identify compatibility issues with the different systems available within the user environment. Acceptance testing also identifies non-functional issues like load and performance defects in the real user environment.

## What Are Pros and Cons?

### Pros

* **Improve Quality**: From the beginning, it ensures that quality is built into the development process, which results in fewer bugs in code and higher-quality software.
* **Reduce Risks**: It provides a clear roadmap for the entire development process, which allows for better risk management and mitigation.
* **Increase Efficiency**: It encourages collaboration between different teams and stakeholders, which results in more efficient development and testing.
* **Improve Communication**: It emphasizes communication between stakeholders, to ensure everyone has a clear understanding of the requirements and objectives.
* **Enhance Testing**: It places a strong emphasis on thorough and effective testing throughout the entire development process.
* **Improve Documentation**: It requires comprehensive documentation at every stage of the development process, which leads to better record-keeping and easier code maintenance.

### Cons

* **Rigid**: It can be inflexible and provide very little room for changes or deviations from the plan. This rigidity can make it difficult to adapt to changing project requirements or new information.
* **Time-Consuming**: It can be time-consuming due to its focus on thorough planning and documentation at every stage. These factors can slow down the development process and lead to longer project timelines.
* **Resource Intensive**: I requires a significant amount of resources including time, budget and personnel, thereby making it a difficult model for small teams or organizations with limited resources to implement.
* **Limited Agility**: It may not be well suited for Agile development approaches, which rely on flexibility, iterative development and continuous feedback.
* **Overemphasis on Testing**: While thorough testing is a critical component of software development, it may place too much emphasis on testing, which can lead to production delays and increased costs.

## When To Use It?

* The software system is **simple and small**.
* The **requirements are well-defined from the beginning**. Also, the project scope is clear, and the development team has a solid understanding of the requirements. In this case, V-model can be an effective tool for delivering high-quality software.

# Scrum Model

Check *Scrum Tutorials.docx*