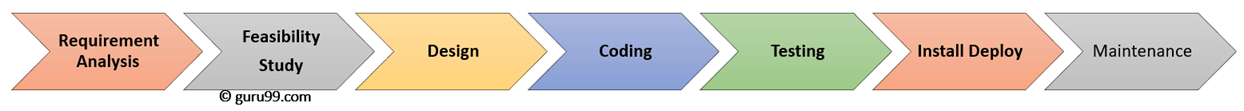
**What is SDLC?**

1. **SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)** is a systematic process for building software that ensures the quality and correctness of the software built.
2. SDLC process aims to produce high-quality software that meets customer expectations.
3. SDLC consists of a detailed plan which explains how to **plan, build, and maintain specific software.**
4. Every phase of the SDLC life cycle has its own process and deliverables that feed into the next phase.

**SDLC Phases**

The entire SDLC process divided into the following stages:



* 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 2: Feasibility study:**

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:**

* SRS is the reference for product architects to come out with the best architecture for the product to be developed.
* Based on the requirements specified in SRS, usually *more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.*
* This DDS is reviewed by all the important stakeholders and based on various parameters **as risk assessment, design modularity, budget and time constraints, the best design approach is selected for the product.**

### Phase 4: Coding:

* Once the system design phase is over, the next phase is coding. In this phase, developers start builds 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.

### Phase 5: Testing:

* During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and sends back to QA for a re-test.

### 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.

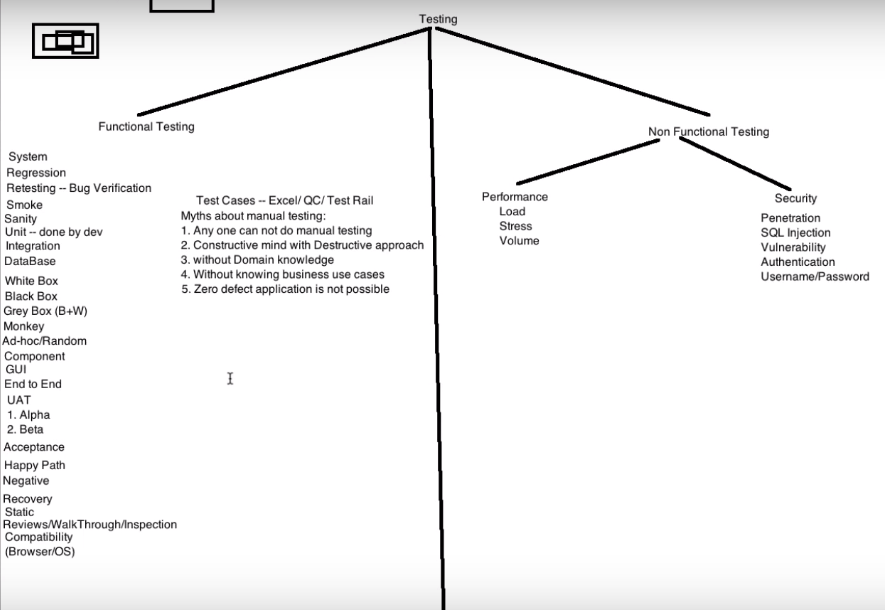
### Phase 7: Maintenance:

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

## Popular SDLC models

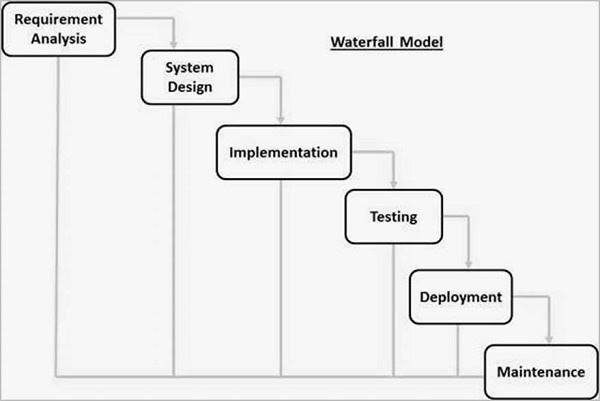
1. Waterfall model
2. Incremental Approach
3. V-model
4. Agile model
5. Spiral Model
6. Big bang model



Waterfall Model – Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

* Requirement Gathering and analysis − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
* System Design − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
* Implementation − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
* Integration and Testing − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
* Maintenance − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

* Requirements are very well documented, clear and fixed.
* Product definition is stable.
* Technology is understood and is not dynamic.
* There are no ambiguous requirements.
* Ample resources with required expertise are available to support the product.
* The project is short.

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.

**But what exactly is the role of QA in agile?**

What can a tester do to help initiate a cooperative working relationship with the development team? Here are 6 things software testers should do when working with an agile scrum team:



**1. Attend sprint-planning sessions.**

A member of QA should always attend planning sessions. This ensures QA is synchronized with the development team from the start, and allows QA to identify possible problem areas and risks early on. Just like developers estimate the effort it will take for them to write code, during the planning session QA should estimate the effort required for testing the code. When QA is left out of the planning session, testing effort / time is often overlooked and not included in the sprint’s overall estimates.

**2. Attend daily stand ups.**

A member of QA should attend daily stand ups. This promotes a collaborative team environment, making QA feel involved and a part of the team. Additionally, by QA being present, they are able to stay up to date with how the sprint is going, which allows them to plan their workload. If a tester has a blocker, they can bring this up during the standup. QA’s presence in stand ups also gives them a chance to give an update on known issues, which in turn, allows developers to keep up to speed on testing progress and plan their own workload.

**3. Don’t save all the testing for the end; test throughout the sprint.**

This is important. In order to deliver high quality software in a short amount of time, you need to work with efficiency. With testing happening throughout the entire duration of the sprint, the workload for QA is spread out, which allows for issues to be found earlier instead of only at the end of the sprint. If you find all the bugs at the end of the sprint, it’s too late. By integrating testing and development, it allows the two teams to work together and resolve issues faster, leading to higher quality results.

**4. Meet with developers for short, hand-off demonstrations. –** communicate with developer and team for daily progress on work

It’s hard to argue the value of in-person communication. Assuming QA and development work in the same location, schedule a quick face-to-face handoff demonstration for each feature. This allows QA to see exactly how the new feature works, and is a good time for them to ask the developer any questions. These handoffs can also bring to light issues the developer may not have considered yet. These interactions also help shorten the feedback loop between development and QA.

**5. Attend sprint retrospectives.**

If your team isn’t meeting at the end of the sprint to discuss successes and failures, you’re missing out on the opportunity to improve future sprints. No matter how good a team is, there will always be room for improvement. Sprint retrospectives are the opportunity to define weaknesses and determine solutions for them. QA needs to be involved in these discussions to ensure any concerns they have are addressed before the start of the next sprint. For example, maybe a lot of the work was delivered to QA late in the sprint, leading to a rushed testing effort. QA might raise this concern to avoid it happening again, the next time.

**6. Document test cases.- Organize your test cases**

Just because you’re an agile team doesn’t mean you should skip documentation. Documentation is important, especially for QA. Keep your documentation lean, becasue changes are bound to happen, but even minimal documentation can add a lot of value to you and your team. For example, if testers shift from project to project, having some test documentation will help get the new team member up to speed faster. [TestLodge](https://www.testlodge.com/) can help with this – we’ve built a lightweight tool to help you keep your test documents organized.

In summary, agile embraces lean, flexible processes, tools and documentation. The traditional QA model has been refined to a delivery-driven role. This requires a collaborative environment where QA is focused on tasks that result in one thing: quickly delivering a high quality product.