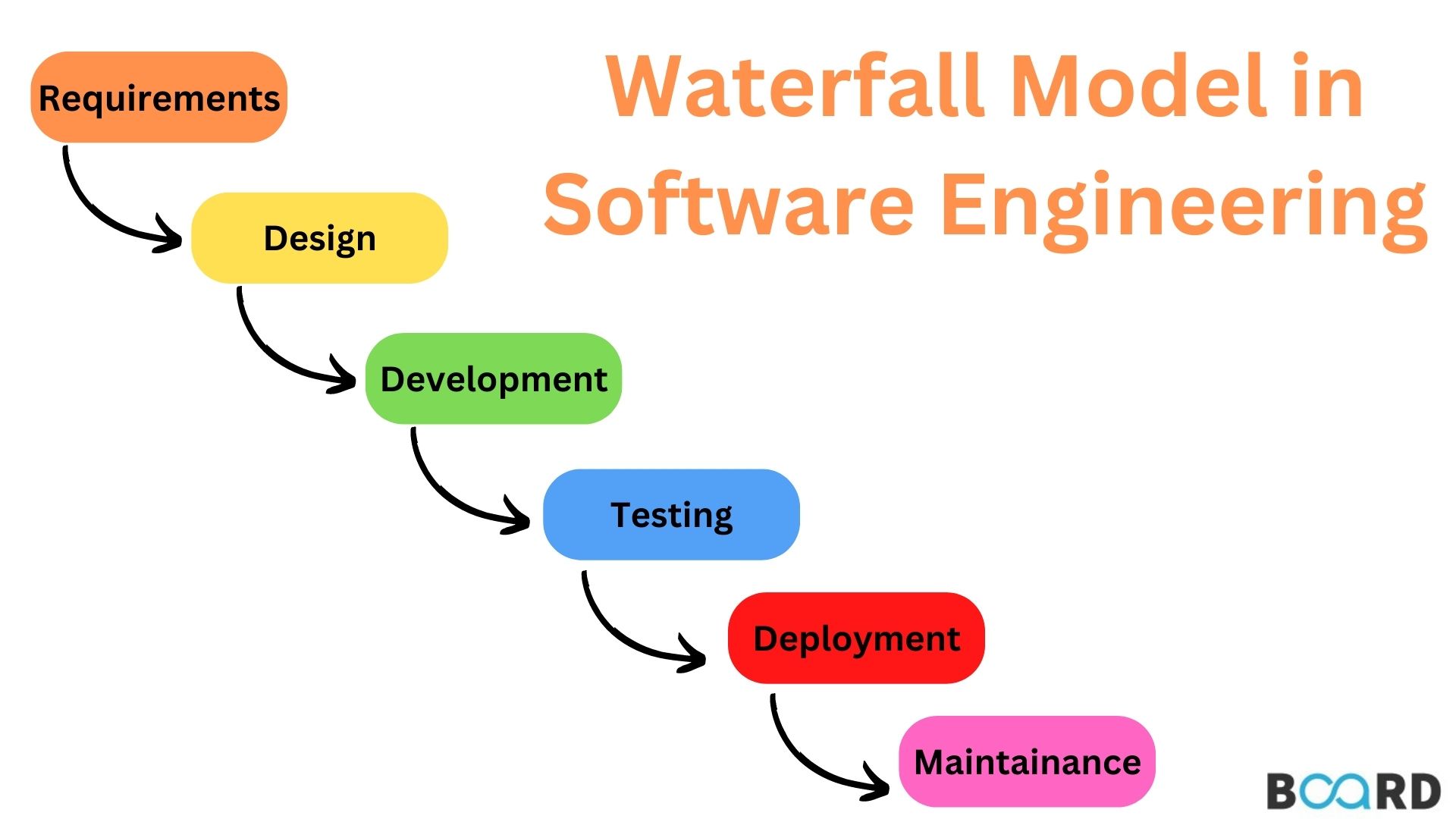
**STLC and QA Testing**

**List down all the Models of SDLC**

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

**Waterfall model in SDLC**



The waterfall is a widely accepted SDLC model. In this approach, the whole process of the software development is divided into various phases of SDLC. In this SDLC model, the outcome of one phase acts as the input for the next phase.

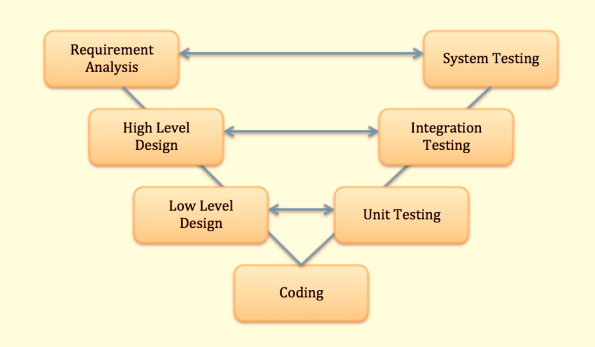
This SDLC model is documentation-intensive, with earlier phases documenting what need be performed in the subsequent phases.

**Incremental Model in SDLC**

The incremental model is not a separate model. It is essentially a series of waterfall cycles. The requirements are divided into groups at the start of the project. SDLC model is followed to develop software. The SDLC life cycle process is repeated, with each release adding more functionality until all requirements are met. In this method, every cycle act as the maintenance phase for the previous software release

For each group, the. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.

**V-Model in SDLC**



In this type of SDLC model testing and the development, the phase is planned in parallel. So, there are verification phases of SDLC on the side and the validation phase on the other side. V-Model joins by Coding phase.

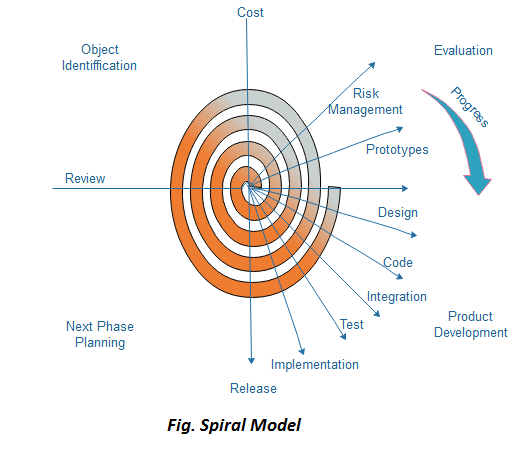
**Agile Model in SDLC**



Agile methodology is a practice which promotes continue interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

**Spiral Model**

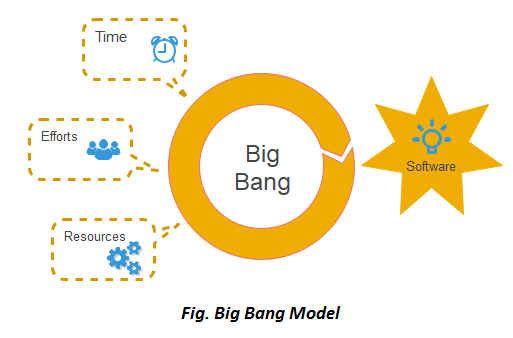
The spiral model is a risk-driven process model. This SDLC testing model helps the team to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc.



This model adopts the best features of the prototyping model and the waterfall model. The spiral methodology is a combination of rapid prototyping and concurrency in design and development activities.

**Big bang model**

Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.



This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements is either unknown or final release date is not given.

|  |  |  |  |
| --- | --- | --- | --- |
| **SDLC Model** | **Advantages** | **Disadvantages** | **Best Use Case** |
| **Waterfall Model** | Easy to understand and use. Clear structure with defined stages. | Cannot move back to a previous stage once it’s done. Doesn’t handle changes well. | Small projects where requirements are very well understood. |
| **V-Shaped Model** | Emphasizes rigorous testing and validation. Clear and simple structure. | Like Waterfall, it’s inflexible to changes. | Projects with clear and fixed requirements, where high reliability is important. |
| **Prototype Model** | Reduces risk of failure, as a working model is seen early. Helps in getting user feedback and refining requirements. | May lead to too much focus on a limited prototype, not the full system. | Projects where user requirements are unclear or complex. |
| **Spiral Model** | High degree of risk management and flexibility. Allows for repeated iterations. | Can be complex to follow/understand. Needs careful management. | Large, complex, and high-risk projects. |
| **Iterative Incremental Model** | Progressive elaboration of the product. Allows for refinement with each increment. | Requires careful planning to make sure increments are meaningful. | Projects where it’s beneficial to get basic functionality out quickly and refine over time. |
| **Big Bang Model** | Minimal planning is required. Can potentially deliver quick results. | High risk, as most work is done simultaneously with minimal to no requirements identified upfront. | Small projects or simple internal tools with one or two developers. |
| **Agile Model** | High flexibility and adaptability. Emphasizes customer satisfaction and team collaboration. | Can be difficult to estimate time and cost. Requires customer and team engagement. | Projects where requirements can change and quick, incremental delivery is desired. |

**2. What is STLC and Explain all Stages of STLC**

Software Testing Life Cycle (STLC) is a procedure for testing software. It involves a sequence of activities that ensure software quality goals are met. The STLC includes both verification and validation activities.

The phases of STLC can be performed multiple times during product development until the product is ready for release. Each phase has entry and exit criteria, activities, and deliverables.

The stages of the STLC include Test Planning, Test Analysis, Test Design, Test Environment Setup, Test Execution, Test Closure, and Defect Retesting. Each of these stages includes specific activities and deliverables that help to ensure that the software is thoroughly tested and meets the requirements of the end users.

The phases of STLC include:

**Test Planning**: Test Planning in STLC is a phase in which a Senior QA manager determines the test plan strategy along with efforts and cost estimates for the project. Moreover, the resources, test environment, test limitations and the testing schedule are also determined. The Test Plan gets prepared and finalized in the same phase.

**Test Planning Activities**

* Preparation of test plan/strategy document for various types of testing
* Test tool selection
* Test effort estimation
* Resource planning and determining roles and responsibilities.
* Training requirement

**Deliverables of Test Planning**

* Test plan/strategy document.
* Effort estimation document.

**Test Case Designing**: The Test Case Development Phase involves the creation, verification and rework of test cases & test scripts after the test plan is ready. Initially, the Test data is identified then created and reviewed and then reworked based on the preconditions. Then the QA team starts the development process of test cases for individual units.

**Test Case Development Activities**

* Create test cases, automation scripts (if applicable)
* Review and baseline test cases and scripts
* Create test data (If Test Environment is available)

**Deliverables of Test Case Development**

* Test cases/scripts
* Test data

**Test Environment Setup**: Test Environment Setup decides the software and hardware conditions under which a work product is tested. It is one of the critical aspects of the testing process and can be done in parallel with the Test Case Development Phase. Test team may not be involved in this activity if the development team provides the test environment. The test team is required to do a readiness check (smoke testing) of the given environment.

**Test Environment Setup Activities**

* Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
* Setup test Environment and test data
* Perform smoke test on the build

**Deliverables of Test Environment Setup**

* Environment ready with test data set up
* Smoke Test Results. **Test Execution:** Real-time validation of product and finding bugs

**Test Closure**: The final stage of STLC

Test Execution Phase is carried out by the testers in which testing of the software build is done based on test plans and test cases prepared. The process consists of test script execution, test script maintenance and bug reporting. If bugs are reported then it is reverted back to development team for correction and retesting will be performed.

**Test Execution Activities**

* Execute tests as per plan
* Document test results, and log defects for failed cases
* Map defects to test cases in RTM
* Retest the Defect fixes
* Track the defects to closure

**Deliverables of Test Execution**

* Completed RTM with the execution status
* Test cases updated with results
* Defect reports

**Defect Retesting**: Evaluate the completion criteria of the software based on test coverage, quality, time consumption, cost, and critical business objectives

Test Cycle Closure phase is completion of test execution which involves several activities like test completion reporting, collection of test completion matrices and test results. Testing team members meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in future, taking lessons from current test cycle. The idea is to remove process bottlenecks for future test cycles.

**Test Cycle Closure Activities**

* Evaluate cycle completion criteria based on Time, Test coverage, Cost,Software, Critical Business Objectives, Quality
* Prepare test metrics based on the above parameters.
* Document the learning out of the project
* Prepare Test closure report
* Qualitative and quantitative reporting of quality of the work product to the customer.
* Test result analysis to find out the defect distribution by type and severity.

**Deliverables of Test Cycle Closure**

* Test Closure report
* Test metrics

**3. As test lead for web based application, your manager has asked to explain the different risk factors that should be included in the test plan**

The most common complaint that comes from software testing using the Agile method is the lack of time. Since the term is itself a synonym for speed, its emphasis on getting things done is self-explanatory. However, this brings up a burning question for every Agile tester in the world.

Most Agile sprints last a couple of weeks. That timeframe is undoubtedly not enough to test every or even most features of modern websites and apps. As development progresses, software becomes more complex and requires more tests to verify its functionality. Running thousands of tests is completely unfeasible, and testers must prioritize what needs to be tested within increasingly shorter timelines.

The answer lies in risk-based testing.

**Risk-based testing:**  If a QA team struggles with deciding how to allocate time and effort in each sprint, their best bet is using risk-based testing. This refers to a testing strategy that uses ‘defined risk’ to determine testing goals. In other words, the risk-based testing approach organizes testing efforts in ways that lower the residual level of product risk when the software goes into production. This strategy is useful for test analysis, planning, estimation, design, execution, and results reporting.

**Before creating a plan for risk-based testing, ask the following questions:**

1. What needs to be tested first?
2. How to reduce the testing effort?
3. How many features can be covered with each test cycle?
4. How to decide what not to test?
5. What metrics are required to measure testing success here?

Since risk-based testing categorizes test scenarios based on the impact each risk will have on business success and user experience, start by defining impact. Whichever feature has the greatest impact on customer experience needs to be tested first.

This mode of risk quantification lets testers define the overall impact of each risk and predict how much damage not testing a particular feature can cause. Use the following five levels of impact:

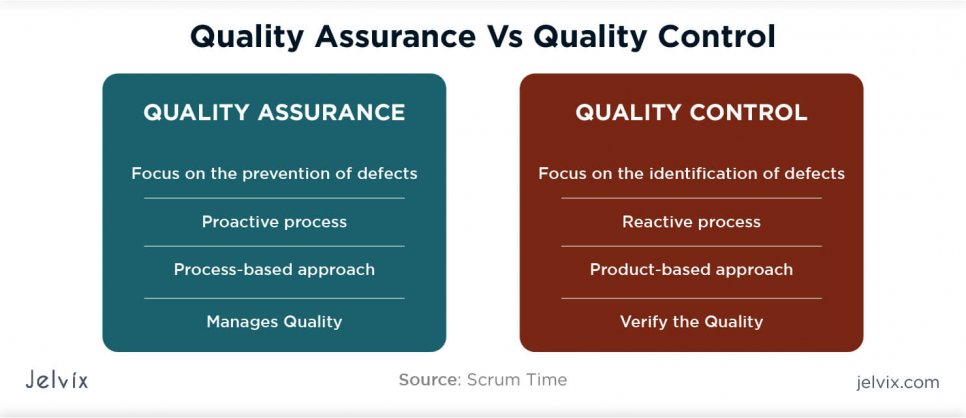
**Critical**: Not testing a feature can harm the software’s central functioning and might lead to revenue loss and lowered credibility.

**Medium**: Not testing a feature may not directly affect customers, but it would still cause significant disruption of the back-end system.

**Moderate:** Not testing a feature will possibly cause some minor inconvenience or annoyance for the customers.

**Marginal**: Not testing a feature will cause little or no disruption. Usually applies only to cosmetic errors.

4. **Quality Assurance and Quality Control**



**Quality Assurance**

Software quality assurance is (also known as QA) a sequence of tasks to prevent defects and ensure that the techniques, methods, approaches, and processes are designed for a specific application must be implemented correctly. This is an ongoing process within the development of a software system.

The development of units of an application is checked under the quality assurance specifications in the sequence of their development.

Quality assurance test ensures the development of high-quality software because of its main focus on the high-quality processes, good quality management system and periodic conformance audit during the development of software. It is a managerial tool includes planned and systematic activities and documentation to prevent problems related to quality.

The responsibility of quality assurance is not of any specific team, but it is a responsibility of each member of the development team.

1. Quality assurance prevents defects.
2. Quality assurance is process oriented.
3. Quality assurance is proactive in a process and preventive in nature.
4. Quality assurance is a managerial tool.
5. Each developer is responsible for quality assurance.

**Quality Control**

Quality Control also known as QC is a sequence of tasks to ensure the quality of software by identifying defects and correction of defects in the developed software. It is a reactive process, and the main purpose of this process is to correct all types of defects before releasing the software. The process is done by eliminating sources of problems (which cause to low the quality) through the corrective tools so that software can meet customer's requirements and high quality.

The responsibility of quality control is of a specific team which is known as a testing team that tests the defects of software by validation and corrective tools.

1. Quality Control provides identification of defects.
2. Quality Control is product oriented.
3. Quality Control is a corrective tool.
4. Testing team is responsible for Quality control.
5. Quality Control is a reactive process.

**Difference between Quality Assurance and Quality Control**

Quality assurance (QA) and quality control (QC) are both responsible for ensuring products meet quality standards. QA focuses on preventing defects, while QC focuses on identifying defects.

| **Quality Assurance** | **Quality Control** |
| --- | --- |
| It is a process which deliberates on providing assurance that quality request will be achieved. | QC is a process which deliberates on fulfilling the quality request. |
| QA aim is to prevent the defect. | A QC aim is to identify and improve the defects. |
| QA is the technique of managing quality. | QC is a method to verify quality. |
| QA does not involve executing the program. | QC always involves executing the program. |
| All team members are responsible for QA. | Testing team is responsible for QC. |
| QA Example: Verification | QC Example: Validation. |
| QA means Planning for doing a process. | QC Means Action for executing the planned process. |
| Statistical Technique used on QA is known as Statistical Process Control (SPC.) | Statistical Technique used on QC is known as Statistical Quality Control (SPC.) |
| QA makes sure you are doing the right things. | QC makes sure the results of what you've done are what you expected. |
| QA Defines standards and methodologies to followed in order to meet the customer requirements. | QC ensures that the standards are followed while working on the product. |
| QA is the process to create the deliverables. | QC is the process to verify that deliverables. |
| QA is responsible for full software development life cycle. | QC is responsible for [software testing life cycle.](https://www.softwaretestinghelp.com/what-is-software-testing-life-cycle-stlc/) |

1. **Difference between manual testing and automated testing**

The difference between manual testing and automated testing is that in manual testing, you perform the tests step by step without the help of tools, whereas in automated testing, tests are executed automatically using automation tools & frameworks. Automation Testing is faster than a manual testing.

Both manual and automation testing approaches have their significant place in the SDLC. The choice between them depends on various factors, including project requirements, time constraints, budget, system complexity, and the testing team’s skills and expertise as listed below.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Manual Testing** | **Automation Testing** |
| **Accuracy** | Manual Testing shows lower accuracy due to the higher possibility of human errors. | Automation Testing depicts a higher accuracy due to computer-based testing eliminating the chances of errors. |
| **Testing at Scale** | Manual Testing needs time when testing is needed at a large scale. | Automation Testing easily performs testing at a large scale with the utmost efficiency. |
| **Turnaround time** | Manual Testing takes more time to complete a testing cycle, and thus the turnaround time is higher. | Automation Testing completes a testing cycle within record time; thus, the turnaround time is much lower. |
| **Cost Efficiency** | Manual Testing requires more cost as it involves hiring expert professionals. | Automation Testing saves costs incurred as once the software infrastructure is integrated; it works for a long time. |
| **User Experience** | Manual Testing ensures a high-end User Experience to the software’s end user, as it requires human observation and cognitive abilities. | Automation Testing cannot guarantee a good User Experience since the machine lacks human observation and cognitive abilities. |
| **Areas of Specialization** | To exhibit the best results, manual Testing should be used to perform Exploratory, Usability, and Ad-hoc Testing. | Automation Testing should be used to perform [Regression Testing](https://www.browserstack.com/guide/regression-testing), Load Testing, [Performance Testing](https://www.browserstack.com/guide/performance-testing), and Repeated Execution for best results. |
| **User Skills** | Users must be able to mimic user behaviour and build test plans to cover all the scenarios. | Users must be highly skilled at programming and scripting to build test cases and automate as many scenarios as possible. |