Testing

Role of a Test engineer

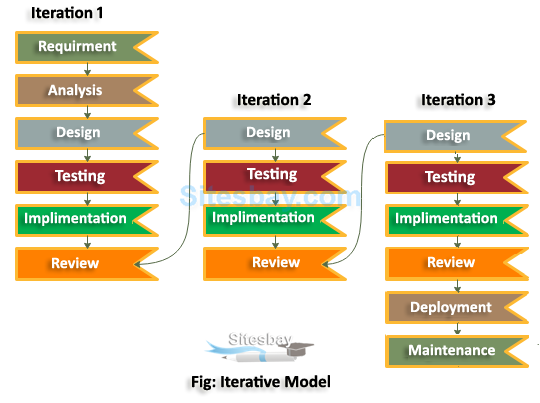
The role of a test engineer, also known as a quality assurance (QA) engineer or software tester, is crucial in the software development lifecycle. Test engineers are responsible for designing and executing tests to ensure the quality, reliability, and performance of software applications or systems. Their primary goal is to identify defects and issues before the software is released to end-users. Here are some key responsibilities and activities typically associated with the role of a test engineer:

1. Test Planning: Test engineers collaborate with stakeholders, such as developers, business analysts, and project managers, to understand requirements and define test objectives, strategies, and plans. They identify test scenarios and determine the scope and depth of testing.
2. Test Design: Test engineers create detailed test cases and test scripts based on the test plan and requirements. They define the expected outcomes and specify the test data and environment needed to execute the tests effectively.
3. Test Execution: Test engineers perform various types of testing, including functional testing, regression testing, performance testing, security testing, and usability testing. They execute test cases, record test results, and compare actual outcomes with expected results. They log defects and work with the development team to resolve issues.
4. Test Automation: Test engineers develop and maintain automated test scripts using testing frameworks and tools. Automation helps improve testing efficiency, accuracy, and repeatability. They identify suitable test cases for automation and ensure that automated tests are integrated into the continuous integration/continuous delivery (CI/CD) pipeline.
5. Defect Management: Test engineers manage the defect lifecycle by logging, tracking, and verifying defects using issue tracking systems. They work closely with developers to investigate and resolve issues, retest fixed defects, and ensure their closure. They also perform root cause analysis to identify underlying causes of defects.
6. Test Documentation: Test engineers create and maintain test artifacts, including test plans, test cases, test data, and test scripts. They document test results, defect reports, and any relevant information related to the testing process. Well-documented test artifacts aid in knowledge transfer and serve as references for future testing efforts.
7. Collaboration and Communication: Test engineers collaborate with cross-functional teams, including developers, business analysts, and project managers, to ensure effective communication and alignment throughout the software development process. They participate in meetings, provide status updates, and share testing progress and findings with stakeholders.
8. Continuous Improvement: Test engineers continuously evaluate and enhance testing processes and methodologies. They identify areas for improvement, propose new strategies, and adopt best practices to enhance the overall quality of the software. They stay updated with the latest testing trends, tools, and technologies to incorporate them into their testing approach.

In summary, test engineers play a critical role in ensuring the quality and reliability of software applications. By conducting thorough testing and identifying defects, they contribute to the overall success of software development projects and help deliver high-quality software to end-users.

SDLC

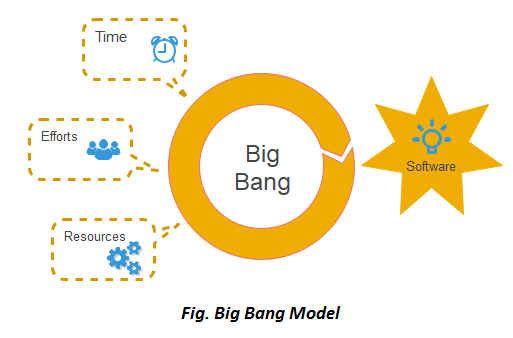
Iterative model



In this Model, you can start with some of the software specifications and develop the first version of the software. After the first version if there is a need to change the software, then a new version of the software is created with a new iteration. Every release of the Iterative Model finishes in an exact and fixed period that is called iteration.

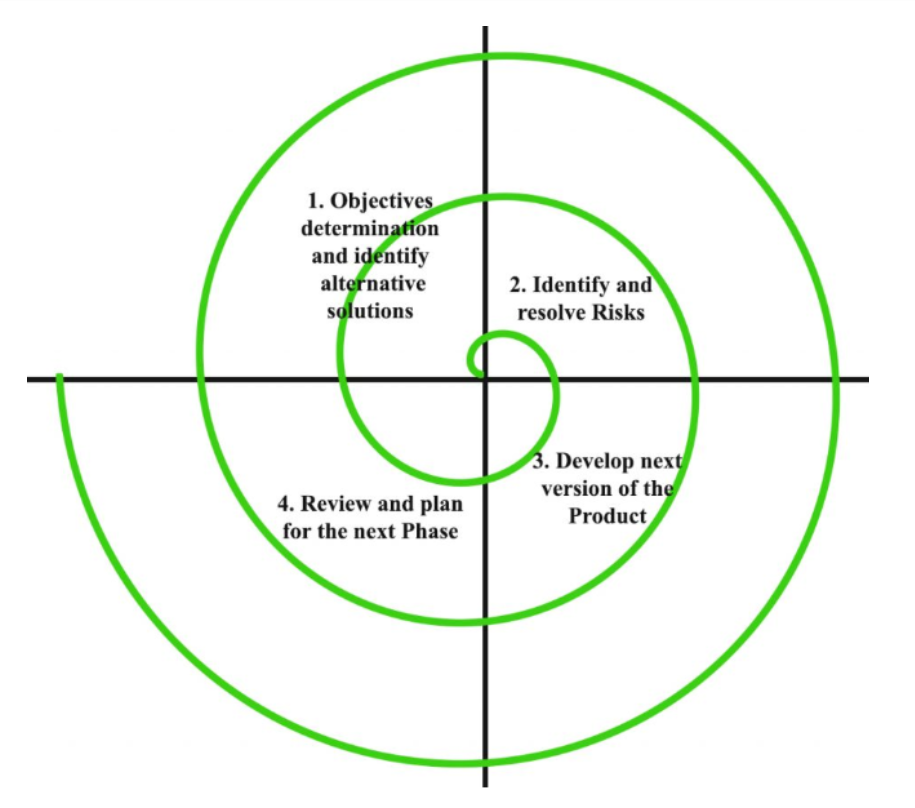
The Iterative Model allows the accessing earlier phases, in which the variations made respectively. The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.

Big Bang Model



The Big Bang model, in the context of software development, is an approach where all the phases of the Software Development Life Cycle (SDLC) are performed simultaneously without following a specific sequence. It is an unconventional and high-risk model that involves a "big bang" approach of developing and implementing the entire software system in one go.

Spiral Model



The Spiral model is a software development life cycle (SDLC) model that combines elements of both the waterfall model and iterative development. It is a flexible and risk-driven approach that allows for iterative development and continuous risk analysis throughout the project. The Spiral model consists of a series of iterative cycles, each involving four key phases:

1. Planning: In this phase, project objectives, requirements, and constraints are identified. The project's scope is defined, and initial plans are made to address various aspects such as budget, schedule, resources, and potential risks. A feasibility analysis is also conducted to determine the project's viability.
2. Risk Analysis: The second phase focuses on identifying, analyzing, and mitigating risks associated with the project. Potential risks are evaluated based on their impact and likelihood, and strategies are developed to address them. This phase helps in decision-making regarding whether to proceed with the project or take necessary corrective actions.
3. Engineering and Development: In this phase, the actual development and implementation of the software take place. Requirements are gathered, design options are explored, and prototypes or small-scale versions of the system are created. The software is developed incrementally and iteratively, with each iteration adding new functionality or improving existing features.
4. Evaluation and Planning: The final phase of each cycle involves evaluating the results of the previous cycle, reviewing the progress, and planning for the next cycle. Feedback is gathered from users, stakeholders, and other project members to assess the software's performance and identify areas for improvement. Based on this evaluation, plans are refined, and the next iteration of the spiral begins.

The key characteristic of the Spiral model is its emphasis on risk management and continual evaluation throughout the software development process. The model allows for flexibility and adaptation to changing requirements, as well as early identification and mitigation of potential risks. Here are some notable features and benefits of the Spiral model:

1. Risk-driven Approach: The Spiral model focuses on identifying and addressing risks early in the development process. This helps in managing and reducing project risks effectively.
2. Flexibility: The iterative nature of the Spiral model allows for flexibility and accommodates changes in requirements. Each iteration builds upon the previous one, incorporating feedback and adapting to evolving needs.
3. Incremental Development: The Spiral model supports the development of software in increments or iterations. This allows for early delivery of functional components, facilitating early user feedback and faster time-to-market.
4. Continuous Evaluation: The model promotes ongoing evaluation and feedback loops. This enables constant monitoring of project progress, quality, and adherence to objectives, allowing for timely adjustments and course corrections.
5. Client Involvement: The Spiral model encourages client involvement throughout the development process. Regular feedback and participation of stakeholders lead to better alignment with user expectations and higher customer satisfaction.
6. Phased Approach: While the Spiral model incorporates elements of iterative development, it still maintains a structured approach similar to the waterfall model. This makes it suitable for projects that require a balance between flexibility and disciplined planning.

The Spiral model is particularly well-suited for complex projects that involve significant risks and uncertainties, as well as projects with evolving or unclear requirements. It provides a systematic approach to manage risks, handle changes, and deliver high-quality software in a controlled manner. However, it requires experienced project management and a diligent focus on risk analysis to ensure its effectiveness.

V Model



**Verification:** It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

**Validation:** It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus it is known as V-Model.

**There are the various phases of Verification Phase of V-model:**

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

**There are the various phases of Validation Phase of V-model:**

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client?s business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

When to use V-Model?

* When the requirement is well defined and not ambiguous.
* The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
* The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

Advantage (Pros) of V-Model:

1. Easy to Understand.
2. Testing Methods like planning, test designing happens well before coding.
3. This saves a lot of time. Hence a higher chance of success over the waterfall model.
4. Avoids the downward flow of the defects.
5. Works well for small plans where requirements are easily understood.

Disadvantage (Cons) of V-Model:

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. Software is developed during the implementation stage, so no early prototypes of the software are produced.
4. If any changes happen in the midway, then the test documents along with the required documents, has to be updated.