# **Software Development Life Cycle (SDLC)**

The Software Development Life Cycle (SDLC) is a process that is used by software development teams to design, develop, test, and deploy software applications. The SDLC consists of several stages, each of which involves specific activities and deliverables. The following are the stages of the SDLC:

1. **Requirements gathering:** In this stage, the development team meets with the client or the end-users to understand their needs and requirements for the software application. The team gathers information about the functionalities, features, and performance expectations of the software application. The deliverable for this stage is the **requirements document**.
2. **Design:** In this stage, the development team designs the software application's architecture and creates a detailed design document. The design document includes the software application's structure, modules, algorithms, data structures, and user interface. The deliverable for this stage is the **design document**.
3. **Development:** In this stage, the development team starts coding the software application according to the design specifications. They use programming languages, frameworks, and tools to develop the software application. The deliverable for this stage is the **source code**.
4. **Testing:** In this stage, the development team performs different types of testing to ensure the software application works as expected. They conduct unit testing, integration testing, system testing, and acceptance testing. The deliverable for this stage is the **test plan and test results.**
5. **Deployment:** In this stage, the development team deploys the software application to the production environment. They configure the hardware and software resources, install the software application, and perform smoke testing to ensure it works as expected. The deliverable for this stage is the deployed **software application**.
6. **Maintenance:** In this stage, the development team maintains the software application by fixing bugs, updating features, and improving performance. They also provide technical support to end-users to resolve issues with the software application. The deliverable for this stage is the **maintenance plan and maintenance reports**.

**Example 1 -** you have been hired by a company to develop a new online shopping website. SDLC would work in this case:

1. **Planning:** In this phase, you would work with the client to gather requirements, identify project objectives, and plan out the project scope, timelines, and budget. You might create a project plan or a scope document that outlines the overall plan for the project.
2. **Analysis:** In this phase, you would analyse the requirements and come up with a detailed functional specification for the website. You might create wireframes or mock-ups to give the client an idea of what the website will look like and how it will function.
3. **Design:** In this phase, you would create a detailed design for the website, including the layout, colour scheme, typography, and overall look and feel. You might also create a style guide or design document to ensure consistency throughout the website.
4. **Development:** In this phase, you would build the website using a programming language and a content management system (CMS). You would also perform unit testing to ensure that each component of the website is functioning correctly.
5. **Testing:** In this phase, you would perform user acceptance testing (UAT) to ensure that the website is working as expected and meets the client's requirements. You might also perform load testing to ensure that the website can handle a high volume of traffic.
6. **Deployment:** In this phase, you would deploy the website to a production environment, making it available to the public. You might also perform post-deployment testing to ensure that everything is still functioning as expected.
7. **Maintenance:** In this final phase, you would maintain the website, performing updates and patches as needed to ensure that it continues to function correctly and meet the client's needs.

So, that's how the SDLC would work for a website development project.

**Example 2 -** SDLC in the context of embedded systems development in the automotive domain.

The development of embedded systems for automotive applications is highly complex and challenging. These systems are designed to provide various functionalities like safety-critical systems, infotainment systems, engine control units, etc. The SDLC for embedded systems development in the automotive domain typically includes the following phases:

1. **Requirements gathering:** In this phase, the project requirements for the embedded system are gathered, which includes **both functional and non-functional requirements**. Functional requirements define what the system must do, while non-functional requirements specify **system performance and quality characteristics, such as reliability, safety, and security.** In the automotive domain, compliance with safety standards such as ISO 26262 is essential.
2. **System architecture and design:** In this phase, the **high-level system architecture** and design are developed, including hardware and software components, interfaces, and algorithms. The **design should consider various factors, such as power consumption, heat dissipation, and weight.**
3. **Software design and implementation:** In this phase, the software modules are designed and implemented based on the architecture and design. The development team uses programming languages and development tools specific to the automotive domain, such as **C, C++, and AUTOSAR (Automotive Open System Architecture), to implement the software.**
4. **Integration and testing:** In this phase, the software components are integrated into the hardware, and the system is tested using various testing methods. Integration testing verifies that the individual software modules work correctly when integrated with other modules, while system testing verifies the system functionality and performance.
5. **Validation and verification:** In this phase, the system is validated and verified to ensure that it meets the requirements and standards. Validation testing is performed to ensure that the system meets the user's needs, while verification testing is performed to ensure that the system meets the defined standards.
6. **Production deployment:** In this phase, the system is deployed into the production environment. At this stage, the system is tested again to ensure that it works as expected in the target environment.
7. **Maintenance and support:** In this final phase, the system is maintained and supported throughout its lifecycle. This includes software updates, bug fixes, and hardware maintenance.

**Initial Capability Document (ICD) & Capability Development Document (CDD)**

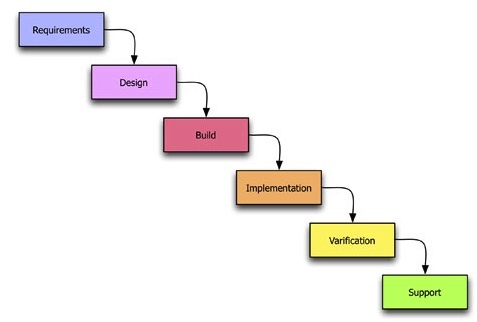
The Initial Capability Document (ICD) and Capability Development Document (CDD) are two important documents that can be used in the System Development Life Cycle (SDLC) to describe the capabilities required for a system or software being developed.

* The **ICD is typically developed during the Conceptualization phase of the SDLC and outlines the basic operational and performance requirements for the system or software being developed.** It provides an **overview of the goals and objectives of the project**, the key **stakeholders**, and the high-level requirements that need to be met to achieve the project's objectives.
* The **CDD, on the other hand, is typically developed during the Design and Implementation phases of the SDLC and provides more detailed and comprehensive information about the system or software being developed**. The CDD includes **information about the operational, performance, and support requirements for the system**, as well as the associated cost and schedule. It provides a blueprint for the system development and serves as a framework for measuring progress and success.

Both **the ICD and CDD are important documents that can be used to guide the development of a system or software in the SDLC.** They help to ensure that the project is focused on meeting the needs of the stakeholders, and that the system or software being developed meets the requirements and specifications outlined in the documents.

# **The Waterfall Model**

The Waterfall Model is **a linear sequential software** development process model that follows a systematic approach to develop software. It consists of a series of phases that must be completed in order, with each phase being completed before the next one begins.



The five phases of the Waterfall Model are:

1. **Requirements Gathering and Analysis:** In this phase, the requirements for the software are gathered and analysed. This involves meeting with stakeholders and end-users to identify their needs and requirements and documenting them in a requirements specification document. This document serves as the foundation for the rest of the development process.
2. **Design:** In this phase, the software design is created based on the requirements specification document. The design specifies the overall architecture of the software, including its modules and components, and how they will interact with each other. The design phase also includes creating a detailed plan for how the software will be implemented.
3. **Implementation:** In this phase, the software is actually developed based on the design. The code is written and compiled, and the software is tested to ensure that it meets the requirements specified in the requirements specification document.
4. **Testing:** In this phase, the software is tested to ensure that it is working as intended and meets the requirements specified in the requirements specification document. This includes functional testing, performance testing, and other types of testing to ensure that the software is working properly.
5. **Maintenance:** Once the software has been tested and deployed, it enters the maintenance phase. In this phase, any bugs or issues that are discovered are fixed, and updates or new features may be added. The maintenance phase continues for the life of the software, as long as it is in use.

One of the key advantages of the Waterfall Model is that it is a well-defined process that is easy to understand and follow. It also ensures that each phase of development is completed before moving on to the next phase, which can help to reduce the risk of errors and bugs. However, the Waterfall Model can be inflexible, and it may not be well-suited for complex or rapidly changing projects.

**Some of the major advantages of the Waterfall Model are as follows –**

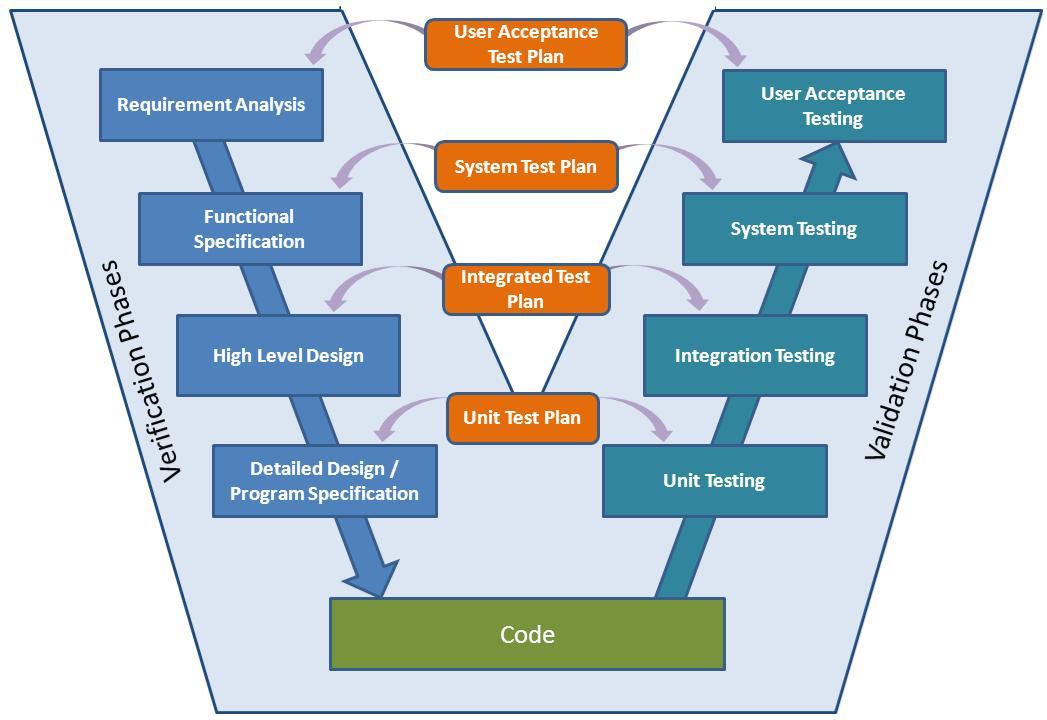
* Simple and easy to understand and use
* Easy to manage, each phase has a specific review process.
* Phases are processed and completed one at a time.
* Works well for smaller projects **where requirements are very well** understood.
* Clearly defined stages.
* Well understood milestones.
* Process and results are well documented.

**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.
* Cannot accommodate changing requirements between the process.
* Adjusting scope during the life cycle can end a project.

# **V-model**

The V-model is a popular software development model that is used to ensure that software systems are developed with a high level of reliability, stability, and quality. It is a sequential process model that outlines the various phases involved in the development of a software system, from requirements gathering to system testing and deployment. The model is called the V-model because the phases and their corresponding testing activities form a V-shape when plotted against a timeline. In this article, we will explore the V-model in more detail, including its various phases and their objectives.



**Phases of the V-model:**

The V-model consists of two main branches, the left-hand branch, and the right-hand branch. The left-hand branch of the V-model represents the planning and design phases of the software development life cycle (SDLC), while the right-hand branch represents the testing and deployment phases. Let's take a closer look at each of these phases.

**Left-hand Branch:**

The left-hand branch of the V-model consists of four main phases:

1. Requirements Gathering and Analysis: In this phase, the requirements for the software system are gathered and analyzed. The requirements gathering process involves interacting with stakeholders and end-users to determine their needs and expectations for the system. This process results in a document that outlines the functional and non-functional requirements for the software system. The requirements analysis process involves reviewing the requirements document to ensure that they are clear, complete, and consistent.
2. System Design: In this phase, the high-level architecture of the system is designed, and the requirements are mapped to the system design. The system design process involves identifying the main components of the system and defining the interfaces between them. This process also involves specifying the system's data structures, algorithms, and control flows. The system design document serves as a blueprint for the development of the software system. **The functional design specifies what the system should do, without specifying how it should be done.**
3. Module Design: In this phase, the detailed design of each module/component of the system is designed, and the module design is mapped to the system design. The module design process involves specifying the data structures, algorithms, and control flows for each module/component of the system. This process also involves identifying the inputs, outputs, and interfaces for each module/component. The module design document serves as a detailed specification for the development of each module/component.
4. Implementation: In this phase, the actual coding of the software system takes place. The implementation process involves translating the design documents into executable code using a programming language. This process also involves testing each module/component as it is developed to ensure that it meets its requirements.

**Right-hand Branch:**

The right-hand branch of the V-model consists of four main phases:

1. Unit Testing: In this phase, each module/component of the system is tested in isolation to ensure that it meets its requirements. The unit testing process involves writing test cases for each module/component and executing them using a testing framework. This process also involves debugging and fixing any defects found during testing.

Unit testing is typically carried out by the developers themselves, using automated testing tools or manual testing techniques. The goal of unit testing is to identify and fix defects as early as possible in the software development life cycle.

1. Integration Testing: In this phase, the modules/components are integrated and tested together to ensure that they work together as expected. The integration testing process involves combining the modules/components and testing them as a group. This process also involves testing the interfaces between the modules/components and ensuring that they are working correctly.

Integration testing is typically carried out by a dedicated testing team or by the developers themselves. The goal of integration testing is to identify and fix defects that arise when modules/components are combined and tested together.

1. System Testing: In this phase, the complete system is tested to ensure that it meets its requirements. The system testing process involves testing the entire system as a whole, including all its components and interfaces. This process also involves testing the system under different conditions to ensure that it works correctly in all scenarios.

System testing is typically carried out by a dedicated testing team, separate from the development team. The goal of system testing is to identify and fix defects that arise when the complete system is tested as a whole.

1. Acceptance Testing: In this phase, the system is tested in the user environment to ensure that it meets the user's needs. The acceptance testing process involves testing the system against the user's requirements and expectations. This process also involves testing the system in the user's environment to ensure that it works correctly in the real world.

Acceptance testing is typically carried out by the end-users or a dedicated user acceptance testing (UAT) team. The goal of acceptance testing is to ensure that the system meets the user's needs and is ready for deployment.

**Advantages of the V-model:**

The V-model has several advantages over other software development models. Some of these advantages include:

* **Testing is integrated throughout the development process**, which helps catch defects early and reduces the cost of fixing them.
* The model **emphasizes the importance of requirements gathering and analysis**, which helps ensure that the system meets the user's needs.
* The model provides a clear and structured framework for software development, which helps in project planning and management.
* The model is easy to understand and implement, which makes it suitable for a wide range of software development projects.

**Limitations of the V-model:**

The V-model also has some limitations. Some of these limitations include:

* The model assumes that the requirements are fixed and well-understood, which may not always be the case in practice.
* The model may **not be suitable for complex software development** projects that require a more iterative and incremental approach.
* The model does **not explicitly address the maintenance and evolution of software systems**, which are important aspects of software development.

**Conclusion:**

The V-model is a popular software development model that emphasizes the importance of testing and quality assurance throughout the software development life cycle. The model provides a structured framework for software development that is easy to understand and implement. However, the model also has some limitations and may not be suitable for all software development projects. Overall, the V-model is a valuable tool for ensuring that software systems are developed with a high level of reliability, stability, and quality.

# **Agile process / Agile model / Agile methodology**

* Agile methodology is a popular approach to software development that emphasizes flexibility, collaboration, and rapid iteration. Instead of working on a single, monolithic project for months or even years, the agile approach breaks the project down into smaller, more manageable pieces called "user stories."
* A user story is a brief, informal description of a specific feature or functionality that the software should provide from the perspective of the end user. For example, a user story might be "As a customer, I want to be able to search for products by keyword so that I can quickly find what I'm looking for." User stories should be written in simple, clear language that everyone on the team can understand, and they should be written from the perspective of the user, rather than from the perspective of the software developer.
* Once the user stories have been identified, the project team will prioritize them based on their importance and complexity. This allows the team to focus on the most important and impactful features first, rather than getting bogged down in less important details.
* The agile methodology also places a strong emphasis on collaboration and communication. Instead of working in isolation, the development team works closely with stakeholders, including customers, business analysts, and product owners, to ensure that the software meets their needs and requirements. This may involve regular meetings, feedback sessions, and demos to showcase progress and gather input.
* The agile methodology also encourages rapid iteration and frequent releases. Instead of waiting until the entire project is complete to release the software, the team releases smaller, more frequent updates that incorporate user feedback and address any issues or bugs that arise along the way.
* Overall, the agile methodology is a flexible, adaptable approach to software development that prioritizes collaboration, communication, and customer satisfaction. By breaking projects down into smaller, more manageable pieces, and working closely with stakeholders throughout the development process, the agile methodology allows teams to deliver high-quality software that meets the needs of its users.
* **Advantages -**
  1. Customer no need to wait for long time.
  2. We develop, test and release a piece of software to the customer with a small number of features. Releases will be fast (weekly). Customer no need to wait for a long time.
  3. We **can accept/accommodate requirement changes** at any time of the software development process.   
     (The customer can provide changes at any time you should be able to accommodate them in our software)
  4. It is a quite easy model to adopt.
  5. There will be **effective communication between Customer & Teams.**
* **Disadvantage –**

Less focused on design and documentation since we deliver software very fast.

* Manifesto for Agile Software Development (The 4 Values and 12 Principles)
* Agile is a broader methodology that emphasizes flexibility, collaboration, and rapid iteration. It is based on the Agile Manifesto, a set of values and principles for software development that prioritize working software, customer collaboration, and responding to change.
* **Scrum**, on the other hand, is a specific framework for implementing the agile methodology. It defines a set of roles, processes, and artifacts that help teams to manage and deliver complex projects.

# **Scrum**

* Scrum is the most widely used framework under the Agile umbrella.
* Agile is a defined process which says we must follow some principle. Scrum says how to follow, how to implement those principles.
* Scrum is a Framework through which we built a software product by following Agile principles.

**3 Principles:**

The following principles underpin the empirical nature of scrum:

* **Transparency**  
  The team must work in an environment where everyone is aware of what issues other team members are running into. Teams surface issues within the organization, often ones that have been there for a long time, that get in the way of the team’s success.
* **Inspection**  
  Frequent inspection points built into the framework to allow the team an opportunity to reflect on how the process works. These inspection points include the Daily Scrum meeting and the Sprint Review Meeting.
* **Adaptation**  
  The team constantly investigates how things are going and revises those items that do not seem to make sense.

**5 Scrum Values:**

The scrum framework believes in cheering the core 5 scrum values. These values are needed to make the scrum process, events, and team successful.

* **Focus**
  + Everyone must focus on the team's efforts and sprint backlog tasks.
  + When we focus on the key items, we deliver the most valuable items at a faster rate.
  + The team must focus on the work done list by the end of every sprint.
  + The team must focus on the scrum goals which tell us what to deliver to make the projects successful.
* **Courage**
  + The team members must have the audacity to do the right things in the projects by solving complex and unpredictable problems. We must show courage to support each other in the team.
  + Develop the courage to share the information and feedback with a reflection on our work to all the team members and stakeholders.

Diagram

Description automatically generated

* **Commitment**
  + The scrum team members must show commitment towards working in a team for achieving the scrum goals by creating realistic goals and not feel satisfied with their achievements but also create team success by encouraging trust, high team standards, and productive problem-solving
  + The scrum master commits by solving the problems and issues of the team which they are unable to solve and removes the impediments along the scrum path.
  + The product owner commits by making the best decisions to optimize the value of the product backlog items.
* **Respect**
  + We need to show respect for all the team members and cultivate and productive, human, and engaging environment for all
  + The entire Scrum team must promote respect for each other's, roles, accountabilities, and diverse perspectives by attending the scrum review, the scrum planning, and Scrum retrospectives.
* **Openness**
  + Have the openness of work, such as the progress made, learning and solve the problems. The team must be open to collaborate and communicate with a wider range of teams, across skills and disciplines
  + By having the openness in changing the way we work, provide feedback, and our reflection the sprint retrospectives focus on continuous improvement.

**Main roles in scrum:**

1. Product owner
2. Scrum master
3. Development team
   * **Product owner** 
     + The product owner is responsible for managing the product backlog to achieve the desired outcome that a product development team seeks to accomplish.
     + He talks to the customer, and he will get features from the customers
     + Define the features of the product.
     + Prioritize features according to market value.
     + Adjust features and priority The functional design specifies what the system should do, without specifying how it should be done.) as needed
     + Accept or reject work results
     + Product owner is a vital member of an Agile or Scrum team. They decide what the final vision and functionality of the product will be. This is based on the demands and requirements of the customer.
   * **Scrum master**
     + The Scrum Master is responsible for ensuring the team lives agile values and principles and follows the processes and practices that the team agreed they would use. (Key role is facilitating and driving the Agile process)
     + The responsibilities of this role include:
       - Clearing obstacles
       - Establishing an environment where the team can be effective
       - Addressing team
       - Ensuring a good relationship between the team and product owner as well as others outside the team
       - Protecting the team from outside interruptions and distractions.
     + Most of the meetings are organized by the scrum master.
   * **Development team/scrum team**

**Agile Scrum Terminology**

**Three artifacts:**  Product Backlog, Sprint Backlog & Product Increment.

**Events/Ceremonies:** Sprint Planning Meeting, Daily Stand-Up Meeting, Sprint Review Meeting, Sprint Retrospective Meeting, Refining the backlog

* **User stories**

A user story is a brief, non-technical, short and simple description of a software requirement written from the customer’s or end-user’s point of view (features)

* **Epic/epic stories**

An epic is a large user story that cannot be delivered as defined within a single iteration or is large enough that it can be split into smaller user stories.

* **Sprint/Iteration/Timebox**

It is fixed length iteration during which an Agile development team produces working software and achieve sprint goals. It may vary from project to project, but an iteration typically lasts between two and four weeks.

* **Daily Stand-up/Daily Scrum meeting/Daily meeting, /Daily huddle**

This is a daily review meeting that is usually hosted by the Scrum Master. This retrospective meeting allows each team member to briefly outline any contributions or issues they may have encountered. Then, the Scrum team plans the day by bringing everyone up to speed. This meeting is usually limited to 15 minutes.

* + The daily meeting is structured around some variant of the following three questions:
  + What have you completed since the last meeting?
  + What do you plan to complete by the next meeting?
  + What is getting in your way?
* **Product Backlog/Backlog**
* A product backlog is the list of requirements requested by the customer. The product backlog is not a ‘to-do’ list; rather, it is a list of all the features the customer has requested be included in the project. The Scrum team uses the product backlog to prioritize features and decide which ones to implement in upcoming sprints.
* It is constantly evolving and is never complete.
* How It’s Used: The product owner is responsible for prioritizing Items in the product backlog, referred to as **Product Backlog Items (PBIs).** The development team pulls the highest-priority PBIs from the product backlog to complete during each sprint. The product owner changes and re-prioritizes the backlog throughout the project development process as needed.
* **Sprint Backlog/ iteration backlog**
* The sprint backlog is a list of everything that the team commits to achieve in each sprint.
* A sprint backlog is the subset/ segment of product backlog items that a team targets to deliver during a sprint to accomplish the sprint goal and make progress toward a desired outcome.
* The sprint backlog also includes any action items the team identified from the previous retrospective meeting.
* The sprint backlog only lasts for the duration of a sprint. Each new sprint starts with a new sprint backlog, although the team may choose to add items from the previous sprint’s sprint backlog to the new sprint backlog.
* **Sprint Planning Meeting**
  + - The Sprint Planning Meeting is a working session held before the start of each sprint
    - The Scrum Master facilitates the meeting while the product owner presents and prioritizes the Product Backlog Items (PBIs) or user stories to be completed by the end of the sprint.
    - The team then breaks down the PBIs or user stories into manageable tasks. Ultimately, the team determines the amount of work they can accomplish during the sprint.
* **Sprint Retrospective**
* A retrospective meeting is a time-boxed meeting that takes place at the end of a sprint, during which the team reflects on the previous sprint and identifies areas for improvement. The focus of the retrospective meeting is on the team's process and how they can work more effectively together. The team discusses what went well, what didn't go well, and what they can do differently in the next sprint to improve their performance. The goal of the retrospective meeting is to identify areas for improvement, to develop action items to address these issues, and to create a plan for implementing these changes in the next sprint.
* **Sprint Review**
* A sprint review meeting, on the other hand, is also a time-boxed meeting that takes place at the end of a sprint, during which the team presents their work to the stakeholders and receives feedback. The focus of the sprint review meeting is on the product that the team has delivered during the sprint. The team demonstrates the working software that they have developed and explains how it meets the requirements and goals of the sprint. The stakeholders provide feedback on the product, ask questions, and make suggestions for future development. The goal of the sprint review meeting is to gather feedback from the stakeholders, to ensure that the product meets their needs, and to plan the next steps for development.
* Retrospective meetings focus on improving the team's process, while sprint review meetings focus on presenting the product to the stakeholders and gathering feedback.

Diagram

Description automatically generated

**Differences between the V-model, Waterfall, and Agile development methodologies:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Methodology | Description | Key Features | Advantages | Disadvantages |
| V-model | A sequential development process that emphasizes verification and validation at each stage of the development cycle. | - Emphasizes testing and quality assurance.  - Each stage must be completed before moving on to the next.  - Testing occurs at each stage, with the goal of identifying and resolving defects early on. | - Emphasizes quality control and assurance.  - Well-suited for projects with clear and stable requirements. | - Limited flexibility to make changes once development has begun.  - Can be time-consuming and expensive.  - May not be suitable for projects with changing requirements. |
| Waterfall | A linear, sequential development process that emphasizes planning and documentation. | - Phases are completed sequentially.  - Each phase must be completed before moving on to the next.  - Emphasizes planning and documentation. | - Emphasizes thorough planning and documentation.  - Well-suited for projects with clear and stable requirements. | - Limited flexibility to make changes once development has begun.  - Can be time-consuming and expensive.  - May not be suitable for projects with changing requirements. |
| Agile | An iterative development process that emphasizes collaboration and flexibility. | - Emphasizes iterative development and continuous feedback.  - Customer involvement throughout the development process.  - Changes and adaptations are expected and welcomed. | - Allows for flexibility and adaptability.  - Well-suited for projects with changing requirements.  - Customer satisfaction is a top priority. | - May require more resources than traditional methods.  - Can be challenging to manage if there is not strong communication and collaboration between team members  - May not be suitable for projects with highly regulated or fixed requirements. |

# **Verification and validation**

Verification and validation are two important concepts in software testing that are often used interchangeably, but they have distinct meanings and purposes.

* **Verification** is the process of ensuring that a software system or component meets its specified requirements and design. It involves reviewing and checking the software code, documentation, and other artifacts to ensure that they conform to the established standards and specifications. Verification answers the question, **"Did we build the product right?"**
* **Validation** is the process of evaluating a software system or component to determine whether it meets the customer's needs and expectations. It involves testing the software in real-world scenarios to ensure that it works as intended and meets the user's requirements. Validation answers the question, **"Did we build the right product?"**

So, verification is about ensuring that the software is built correctly, while validation is about ensuring that the software is the right product for the user. Both verification and validation are critical components of software testing and should be performed throughout the software development lifecycle to ensure the quality and effectiveness of the software product.

**An example** to clarify the difference between verification and validation:

Suppose a company is developing a software system for an online shopping website.

* **Verification:** The software developers will check if the software code meets the requirements and design specifications. For example, **they will check if the code correctly implements the shopping cart feature, calculates taxes and shipping fees accurately, and handles credit card transactions securely.** They may also review the documentation and code review reports to ensure the code quality meets the company's coding standards.
* **Validation:** Once the software system is developed, the company will conduct user **acceptance testing to ensure that the system meets the customer's needs and expectations.** For example, they will test if **users can easily browse and search for products, add items to their shopping cart, checkout with ease, and receive confirmation of their orders. T**hey will also test if the system is secure, stable, and performs well under various user scenarios.

In this example, verification is about ensuring that the software code meets the design and coding standards, while validation is about testing the software to ensure that it meets the user's requirements and expectations.

There are several types of tests related to verification mentioned below:

1. **Unit Testing:** This type of testing focuses on verifying the functionality of individual units of code or modules. It helps to identify defects early in the development process and ensures that each unit of code works as intended.
2. **Integration Testing:** This type of testing verifies that the individual units of code or modules work together as intended. Integration testing can identify defects that arise when the modules are combined.
3. **System Testing:** This type of testing verifies that the entire system works as intended. It ensures that the system meets the requirements and design specifications, and that it can handle expected and unexpected scenarios.
4. **Performance Testing:** This type of testing verifies the system's performance under various loads and stresses. The goal is to ensure that the software system meets the performance requirements and design specifications. This type of performance testing can identify performance bottlenecks, memory leaks, and other issues that can impact the system's performance.
5. **Security Testing:** This type of testing verifies that the system is secure and protected from unauthorized access, attacks, and data breaches. It helps to identify vulnerabilities in the system and ensures that the system can withstand security threats.
6. **White-box testing:** White-box testing is a testing methodology in which the tester has knowledge of the internal workings of the software being tested. This means that the tester has access to the source code and is able to analyse the structure and logic of the software. White-box testing is also known as clear-box testing, open-box testing, or structural testing.
7. **Black-box testing:** Black-box testing is a testing methodology in which the tester has no knowledge of the internal workings of the software being tested. This means that the tester is only concerned with the inputs and outputs of the software, and is not concerned with how the software works internally. Black-box testing is also known as functional testing or closed-box testing.

There are several types of tests related to **validation** mentioned below:

1. **Acceptance Testing:** This type of testing verifies that the software system meets the customer's requirements and expectations. It involves testing the software in real-world scenarios to ensure that it works as intended and meets the user's needs.
2. **Usability Testing:** This type of testing verifies that the software is easy to use and understand. It focuses on the user interface and user experience, and it helps to identify usability issues that could impact user adoption and satisfaction.
3. **Compatibility Testing:** This type of testing verifies that the software works correctly with other systems, devices, and software applications that the user may have. It helps to ensure that the software works seamlessly with the user's environment and does not cause any conflicts or compatibility issues.
4. **Regression Testing:** This type of testing verifies that the software still works correctly after changes or updates are made to the system. It helps to ensure that new features and changes do not cause unexpected issues or conflicts with existing functionality.
5. **Performance Testing:** This type of testing verifies that the software performs well and meets the user's performance requirements. The goal is to ensure that the software system performs well under various real-world scenarios and meets the user's performance expectations. This type of performance testing can identify how the system performs when subjected to the expected and unexpected user loads, stress, and other factors.

Design space exploration is a process of analysing and evaluating different design choices and trade-offs in order to identify the most effective and efficient design solution. It involves exploring a range of design options and assessing their performance and suitability based on various criteria.

The goal of design space exploration is to help designers and engineers make informed decisions about the design of a system or product. It can be used in various fields such as software engineering, mechanical engineering, and electronic design.

* In the context of software engineering, design space exploration involves analysing and evaluating different software architectures and design patterns to identify the most effective and efficient approach for a given problem. For example, a software engineer may explore different database technologies, programming languages, or software frameworks to determine the best approach for developing a new application.
* In electronic design, design space exploration may involve analysing and evaluating different circuit topologies, component choices, and manufacturing processes to optimize performance, reliability, and cost.

Overall, design space exploration is a critical process in the design of complex systems and products, as it allows designers and engineers to make informed decisions and optimize their designs based on various criteria such as performance, cost, and reliability.

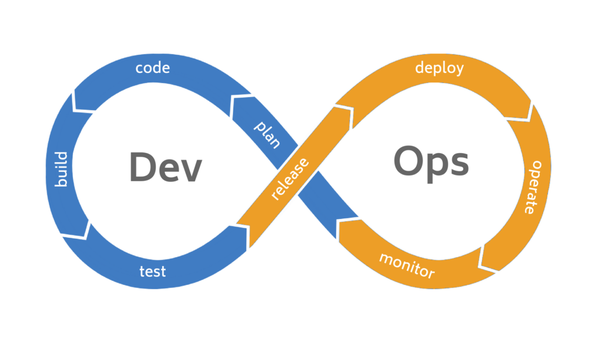
# **Developers and operations teams**

* Developers and operations teams are two key groups involved in the software development process.
* **Developers** are responsible for creating and writing the code that makes up the software. They typically work with programming languages, tools, and frameworks to create the functionality and features that the software needs. Developers are focused on writing high-quality code that is efficient, reliable, and meets the needs of the users.
* **Operations teams**, on the other hand, are responsible for managing the infrastructure and systems that the software runs on. They typically handle tasks such as configuring servers, managing databases, and monitoring the software's performance. Operations teams are focused on ensuring that the software is available, reliable, and performant, and that any issues or incidents are quickly resolved.
* Traditionally, developers and operations teams have worked in separate silos within organizations. This can lead to communication and collaboration issues, as well as delays and inefficiencies in the software development process. **DevOps seeks to bridge the gap between these two groups,** by encouraging collaboration and communication between developers and operations teams, and by using modern tools and processes to streamline the software development and delivery process.
* When developers and operations teams are separate, there can be several issues that arise in the software development process, including:
* **Lack of communication and collaboration:** When developers and operations teams work in separate silos, there is often a lack of communication and collaboration between the two groups. This can lead to misunderstandings, delays, and inefficiencies in the software development process.
* **Slow time to market:** When developers and operations teams work separately, there may be delays in getting software released to the market. This can be due to issues such as miscommunication, lack of alignment on priorities, or inefficient processes.
* **Increased risk of errors and downtime:** When developers and operations teams are separate, there may be a higher risk of errors and downtime in the software. This is because there may be gaps or inconsistencies in the process, and the two groups may not be aligned on priorities or best practices.
* **Lack of visibility and transparency:** When developers and operations teams work separately, there may be a lack of visibility and transparency into the software development process. This can make it difficult to identify issues or bottlenecks and can make it challenging to track progress or measure success.

Overall, when developers and operations teams are separate, there can be a lack of alignment, communication, and collaboration that can lead to delays, errors, and inefficiencies in the software development process. DevOps seeks to address these issues by bringing these two groups together and creating a culture of collaboration, communication, and continuous improvement.

# **DevOps** (**development and operations**)

* **DevOps** is a set of practices that aims to improve collaboration, communication, and automation between software development teams and operations teams to deliver high-quality software products more efficiently. While DevOps has traditionally been associated with web and cloud-based applications, it can also be applied to embedded systems.



* An embedded system is a computer system that is integrated into a larger system, often with specific hardware requirements and constraints. Applying DevOps principles to embedded systems can help teams work more efficiently, reduce errors, and deliver more reliable products.
* **Continuous Integration (CI)** is the practice of automatically building and testing software changes as soon as they are committed to version control. For embedded systems, this can include compiling code, building images, and running unit tests on the target hardware. This ensures that any issues are caught early in the development process, before they become more difficult and expensive to fix.

For example, let's say that a team is developing firmware for an IoT device that collects and analyses data from sensors. With CI, every time a developer commits changes to the codebase, the system automatically builds the firmware and runs tests on a target device. If any issues are found, the team is notified immediately, and they can work to fix the issue before continuing with further development.

* **Continuous Testing (CT)** is the practice of continuously testing software to ensure that it meets requirements and functions correctly. This can include automated testing, manual testing, and exploratory testing. For embedded systems, CT can include testing on the target hardware to ensure that the firmware works as expected.

For example, let's say that a team is developing software for a medical device that monitors patient vital signs. With CT, the team can continuously test the software on the target device to ensure that it meets safety and regulatory requirements. This can include running automated tests to simulate different scenarios, manual testing by medical professionals, and exploratory testing to uncover potential issues.

* **Continuous Delivery (CD)** is the practice of automating the deployment of software changes to production. For embedded systems, this can include deploying firmware updates to devices in the field.

For example, let's say that a team is developing firmware for an automotive system that controls the engine and transmission. With CD, the team can automate the deployment of firmware updates to vehicles in the field, ensuring that all vehicles are running the latest version of the software. This can help to improve reliability and safety, as well as reduce the risk of expensive recalls.

**Applying DevOps principles to embedded systems can help teams work more efficiently, reduce errors, and deliver more reliable products. By using Continuous Integration, Continuous Testing, and Continuous Delivery, teams can catch issues early in the development process, ensure that software meets requirements and functions correctly, and automate the deployment of software changes to production.**

* **Power rails**
* Power rails are **the electrical conductors on a printed circuit board (PCB) that provide power to the various components on the board.** They are typically designed to deliver a specific voltage level to each component on the board and are used to ensure that the components receive the correct amount of power to operate properly.
* Power rails are an essential part of the power distribution system on a PCB and are typically designed to deliver power from a power source, such as a battery or power supply, to the various components on the board. They may also include decoupling capacitors or other components that help to reduce noise or stabilize the power supply voltage.
* The design of power rails is an important consideration in the design of a PCB, as they can affect the performance and reliability of the board. Poorly designed power rails can lead to voltage drops, noise, or other issues that can cause components to malfunction or fail. Therefore, careful attention should be paid to the design of power rails to ensure that they are designed to deliver the correct voltage levels and are properly decoupled and stabilized.
* **Board bring-up**
* Board bring-up is the process of initializing and testing a newly designed printed circuit board (PCB) or electronic device to ensure that it functions properly. This involves connecting the various components on the board, configuring the firmware or software, and performing various tests to verify that the device works as intended.
* During the board bring-up process, engineers typically use specialized tools and equipment to test and verify the functionality of the various components on the board. This may involve using oscilloscopes, logic analyzers, or other testing equipment to verify that the board is working correctly and that data is flowing through the board as expected.
* Board bring-up is a critical step in the development process, as it ensures that the board is working as intended before it is released for further testing or production. It also allows engineers to identify and resolve any issues or problems with the board early in the development process, which can help to reduce development time and costs

# **Limitation & Constraints**

* Limitation is something over which the organization has no control, i.e.- weather, whereas,
* A Constraint is a self-imposed control by the organization, i.e.- for safety reasons we will not work during dark/night-time hours.
* **Common constraints that may occur in an embedded project:**

1. **Resource constraints:** Embedded projects often operate with **limited resources, such as limited memory, processing power, or energy.** These constraints can impact the selection of hardware, software, and development tools.
2. **Time constraints:** Embedded projects often have **strict deadlines** that must be met, such as meeting product release schedules or regulatory requirements. These constraints can impact the development process and the scope of the project.
3. **Cost constraints:** Embedded projects often have **strict budget constraints** that must be adhered to. These constraints can impact the selection of hardware, software, and development tools, as well as the scope of the project.
4. **Safety constraints:** Embedded projects that are used in **safety-critical applications**, such as medical devices, aviation, or automotive systems, **must adhere to strict safety standards and regulations.** These constraints can impact the development process, the selection of hardware and software, and the testing and validation procedures.
5. **Performance constraints:** Embedded projects must often **meet specific performance requirements**, such **as fast response times or high reliability**. These constraints can impact the selection of hardware, software, and development tools, as well as the testing and validation procedures.
6. **Compatibility constraints:** Embedded projects must often be compatible with existing systems or platforms. These constraints can impact the selection of hardware, software, and development tools, as well as the testing and validation procedures.
7. **Environmental constraints:** Embedded projects that are used in harsh environments, such as industrial or military applications, **must** **adhere to specific environmental requirements,** such as **temperature, humidity, or vibration**. These constraints can impact the selection of hardware, software, and development tools, as well as the testing and validation procedures.

# **Some Important Documents:**

* **Requirements Specification Document:**

This document outlines the project's requirements, including functional and non-functional requirements, as well as any constraints or limitations.

* **Design Document:**

This document describes the system architecture, including hardware and software components, interface designs, data flow diagrams, and database schemas.

* **Test Plan Document:**

This document outlines the testing strategy, including test cases, test scenarios, and test scripts. It also defines the testing environment, tools, and resources required.

* **User Manual:**

This document provides instructions on how to use the software, including installation, configuration, and troubleshooting.

* **System Manual:**

This document provides a detailed description of the system's operation, including the system's functionality, design, and maintenance.

* **Deployment Plan Document:**

This document outlines the deployment strategy, including installation and configuration of the software, as well as any necessary hardware or software components.

* **Traceability Matrix:**

This document tracks the requirements throughout the development process and ensures that each requirement is met in the final product.

**Functional requirements** define what the software should do in terms of the system's behavior and functionality. They specify the features, functions, and capabilities of the software that enable it to perform the intended tasks. Examples of functional requirements include **user interfaces, data input and output, calculations, algorithms, and business logic**. Functional requirements are usually **expressed in terms of use cases, user stories, or system requirements specifications.**

**Non-functional requirements**, on the other hand, define how the software should perform in terms of its **quality attributes.** They specify the **software's characteristics, such as reliability, scalability, security, performance, usability, and maintainability.** Non-functional requirements are not directly related to the software's functionality but are essential for ensuring that the software meets the users' needs and expectations. Examples of non-functional requirements include **response time, uptime, data storage capacity, data transfer rates, and error handling**. Non-functional requirements are usually expressed in terms of **quality assurance standards, service level agreements, or performance requirements**.