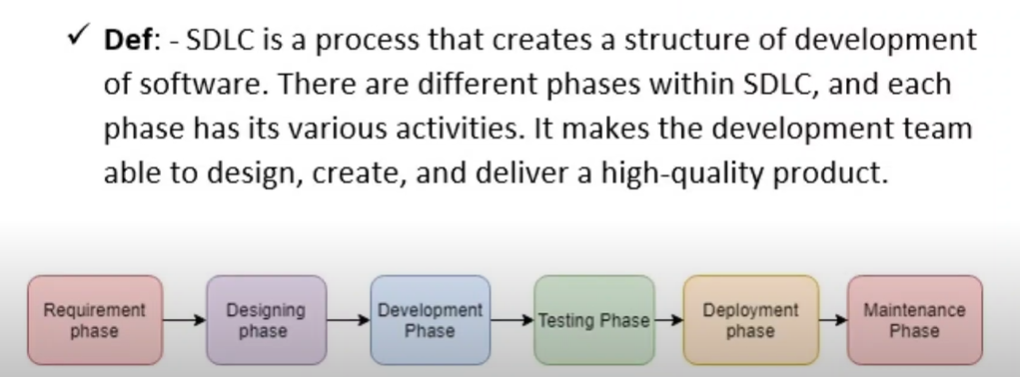
**Introduction.**

Before learning the **DevOps** we should have idea on the **SDLC.**

Now we will see what is **SDLC**.

* SDLC is Software Development Life Cycle it is a process to follow to deliver the product to the end user.
* There are different types of steps we have to follow to deliver the product.



Let us see those steps with one real time example..

Imagine an e-commerce company wants to build a new online platform.

1. **Requirements Phase**: The company defines all the features the website should have, such as user registration, product catalog, shopping cart, payment gateway, and order tracking.
2. **Design Phase**: The development team designs the system architecture, including how the database will store user and product data, how the front end will look, and how the payment gateway will be integrated.
3. **Development Phase**: Developers write code to implement all the features of the website. They build the shopping cart, implement the payment gateway, and create the product catalog.
4. **Testing Phase**: Once development is complete, testers will test the code.
5. **Deployment Phase**: The website is launched to customers.
6. **Maintenance Phase**: Developers start working on improving the search functionality and fixing post-launch issues.

**There are different methodology to follow this SDLC.**

In this we will see the first one that is **Waterfall model.** This is very old model. There are many disadvantages in this methodology.

The Waterfall Model is a linear sequential SDLC model where each phase must be completed before the next one can begin. It's often likened to a waterfall, where the flow of work progresses downwards in a fixed sequence.

**Real-Time Example: Building a House**

Imagine building a house using the Waterfall Model.

1. **Requirement Analysis:** You create a detailed blueprint specifying the number of rooms, materials, and desired features.
2. **System Design:** You design the foundation, walls, roof, and interior layout based on the blueprint.
3. **Implementation:** You start constructing the house according to the design, laying the foundation, building the walls, and installing the roof.
4. **Testing:** You inspect the house for any structural defects or quality issues.
5. **Integration and Deployment:** You complete the interior work (painting, flooring, fixtures) and move into the house.
6. **Maintenance:** You regularly maintain the house (repairs, cleaning, updates).

If you discover a major flaw in the foundation after the walls are built, it would be very difficult and costly to go back and fix it. This illustrates the rigidity and potential risks associated with the Waterfall Model.

There is a lot of time waste in this if development team completes his work then only testing team will test the code until the **testing team has no work.**

**In conclusion,** while the Waterfall Model can be effective for smaller projects with well-defined requirements, its limitations make it less suitable for larger, more complex projects where flexibility and adaptability are crucial. Modern SDLC methodologies like Agile and DevOps have emerged to address the shortcomings of the Waterfall Model and provide more iterative and collaborative approaches.

There are lot of disadvantages in this so we moved to the **AGILE model.**

In this we will see why we have to move from waterfall model to the Agile model with a good real time example.

To explain this shift with a **real-time example** like **constructing a house**, we’ll compare the Waterfall and Agile approaches in this context:

In the **Waterfall Model**, the process of constructing a house would follow strict, sequential steps, with no feedback or changes allowed once each step is completed:

1. **Requirements Gathering (Blueprint)**: You define everything upfront: the number of rooms, layout, materials, design, etc. You finalize a blueprint of the house without getting feedback once the project starts.
   * **Example**: You decide that the house will have 3 bedrooms, 2 bathrooms, a living room, and a kitchen. Once the blueprint is finalized, there is no scope to change it later.
2. **Design**: The architect creates a complete design of the house based on the blueprint, including details like plumbing, electrical wiring, and structural design.
3. **Implementation (Construction)**: Builders start construction according to the finalized design. Once the foundation is laid, no changes can be made to the layout or structure.
4. **Testing (Inspection)**: After the construction is complete, inspectors check the plumbing, electrical systems, and overall structure for safety and functionality.
5. **Deployment (Move-in)**: The house is handed over to the owner once construction is completed and all inspections are done.
6. **Maintenance**: Any issues that arise after moving in, such as a leaky roof or plumbing problems, are fixed in this phase.

**Challenges in the Waterfall Model for House Construction**

* **Rigid Process**: If you realize midway through construction that you want to add a home office or change the position of a room, it’s very difficult and expensive to modify the design. You’d have to tear down completed parts and rebuild.
* **No Customer Feedback Until Late**: You don’t get to see the house or give feedback until the final stages. If you’re unhappy with the result (e.g., the rooms feel smaller than expected), it’s too late to make significant changes without incurring high costs.
* **High Risk**: All decisions are made upfront, and there’s no flexibility to adapt to changes in your needs or lifestyle.

**Agile Approach to House Construction**

In the **Agile Model**, the process is broken into smaller, iterative phases, with continuous feedback and adjustments throughout. Instead of trying to build the entire house in one go, you would **build and review parts of the house incrementally**, adapting the plan as needed.

1. **Iteration 1: Foundation and Framework**:
   * Start by building the foundation and basic framework (the skeleton of the house) without finalizing every detail.
   * **Customer Feedback**: You can walk around the space and see how the layout feels. If you decide that the living room feels too small, the builders can adjust the room sizes or make other structural changes in the next iteration.
2. **Iteration 2: Interior Layout**:
   * The next phase involves adding walls, plumbing, and basic utilities.
   * **Customer Feedback**: Once you see how the space feels with walls up, you might decide that the kitchen should be more open or that you need an extra closet. The builders can make those adjustments without drastically affecting the schedule.
3. **Iteration 3: Finishing Touches**:
   * Focus on adding flooring, painting, electrical fixtures, and cabinetry in this phase.
   * **Customer Feedback**: You can review the colors, finishes, and materials. If you feel a particular paint color looks off, the team can change it before proceeding to the next phase.
4. **Continuous Feedback**: In each iteration, you get a working version of part of the house, provide feedback, and see adjustments happen in real-time. You also have flexibility in design decisions as the house evolves.

In this in every iteration we will follow the SDLC phases.

in Agile, each iteration (often called a **sprint**) follows the phases of the **SDLC** (Software Development Life Cycle), but on a smaller scale. Unlike the Waterfall model, where each SDLC phase is completed once for the entire project, Agile breaks the project into smaller, manageable increments or **iterations**. Each iteration delivers a functional part of the software, and the SDLC is repeated for each increment.

**How the SDLC is Applied in Agile Iterations:**

**1. Planning (Requirements Gathering)**

* At the beginning of each iteration, the development team works with stakeholders to identify and prioritize the requirements (or user stories) that need to be completed during the sprint.
* **Agile Example**: For an e-commerce website, in the first iteration, you might focus on user registration and login functionality. Requirements for these features are gathered, and the team decides what specific tasks to complete in the sprint.

**2. Design**

* Once the requirements for the iteration are clear, the team quickly designs the architecture and technical solutions needed for that increment.
* **Agile Example**: In the e-commerce site’s second iteration, you might work on the product catalog. The design for the database schema, API, and user interface for displaying products is done during this sprint.

**3. Development (Implementation)**

* After the design is finalized for the current iteration, the development team starts writing code to implement the features.
* **Agile Example**: The developers write code to implement the user registration and login features in the first sprint, ensuring that the core functionality is working and ready for testing.

**4. Testing**

* Testing is done concurrently with development in Agile. The QA team tests the features being developed in each iteration to ensure they are functional and meet requirements.
* **Agile Example**: As soon as the login functionality is developed, the QA team runs tests to ensure the login page works, handles invalid logins, and follows security standards (e.g., password encryption).

**5. Deployment**

* At the end of the iteration, the working software (or feature) is delivered, often deployed to a staging or production environment where stakeholders can see the progress.
* **Agile Example**: The login and registration features are deployed to a staging environment where stakeholders can test them and provide feedback before moving on to the next iteration.

**6. Maintenance**

* Each feature is maintained and improved in subsequent iterations. Bugs found after deployment are fixed in future sprints, and new improvements are continually added.
* **Agile Example**: After deploying the login system, you may receive user feedback about issues with the password reset process, which will be addressed in the next sprint.

Yes, each iteration in Agile follows the SDLC process, but rather than applying the SDLC to the entire project all at once (like in Waterfall), Agile applies the SDLC in a **continuous, iterative** manner. Every iteration (or sprint) cycles through the phases of planning, design, development, testing, and deployment to deliver functional software increments.

**Still there are some problems in Agile here DevOps came into the picture.**

The shift from Agile to DevOps isn't about leaving Agile behind but rather enhancing it. Agile focuses on improving software development processes through iterative, collaborative, and incremental approaches. However, as software delivery became more complex, Agile alone wasn't enough to address the entire lifecycle, including deployment and operations. That's where DevOps comes in—bridging the gap between development (Agile) and operations.

**Example**

**Agile**: Imagine you're constructing a house. Agile would be like building the house in phases—first, you lay the foundation, then the walls, roof, and so on. After each phase, you get feedback from the homeowner and make adjustments. The team focuses only on building the structure (development), leaving the final furnishing and utilities (like plumbing and electricity) to another team. While Agile ensures flexibility and quick response to changes, it doesn’t fully consider how everything comes together at the end or how the house will be maintained.

**DevOps**: Now, imagine adding DevOps to this process. Instead of waiting until the house is fully built to install utilities, the construction team works together with the electricians, plumbers, and landscapers from day one. Every phase of construction includes integration with these other teams, ensuring that by the time the walls are up, plumbing and wiring are already in place. Not only is the house built incrementally, but it is also ready for immediate use as soon as it's finished. Furthermore, the same team maintains the house, fixes leaks, and makes upgrades after it’s built.

**Why the Shift?**

In software, Agile helps with fast, incremental development, but it doesn't handle what happens after the code is written—how to deploy, monitor, and scale it. DevOps came in to manage these gaps, adding automation, collaboration across all teams, and continuous deployment, so the software is always ready for release and can be maintained with ease.

**Real-world tech example**: Think of companies like Netflix. They use DevOps to ensure that new features can be deployed quickly and frequently, with automated tests, monitoring, and rollbacks if needed. Without DevOps, new features might take longer to deploy and may face more issues in production due to lack of integration between development and operations teams.

Devops is a culture / Software developement approach , which involves continueosly combing the code , continueosly testing the code ,

continueosly integrating the tools , continueosly deploying the applications, continueosly monitoring the applications through out entire

its software developement life cycle.

===================================================================================================

Continueos Integration :

CI means continueosly developing the code , continueosly combining the code, so that we can fix the bugs as per the time line

and also we can reduce the time during the release in your line.

Continueos Delivery: ( Manual approach)

It is nothing but what ever we generate the package into your environment. suppose If I want continueosly delivery into each and

every environment. If we can do review and then we can deploy into the environment.

continueos Deployment :( with out manual)

CD is peace of practice to design ensure the code can be rapidly or safely deployed into live environment production by delivering each

and every change in the production.

Devops workflow / Jenkins work flow :

1. code commit (github)

2. Build ( maven)(war / ear / jar )

3. code quality ( Sonarqube --->> critical / major / blocker )

4. release package (war / ear / jar )

5. uploading package into Nexus ( Artifactory)

6. deployment ( tomcat ) ---->> dev/qa/uat/prod

7. Test