Maintenance Phase:

- Address bugs, UI issues, and unlisted requirements post-deployment
- Suggest code enhancements
- Incorporate feedback into future software releases, potentially restarting the SDLC process

Conclusion: The SDLC is a structured approach to software development, consisting of six phases that guide the process from planning to maintenance. Each phase has specific tasks and objectives, ensuring a systematic progression toward the development of high-quality software.

Building Quality Software

After reading this lecture, you will be able to:

- List common software engineering processes.
- Describe the common software engineering processes required for building high-quality software.

There are numerous processes that are common to software engineering projects. In this lecture, we will discuss six of them:

1. **Requirements Gathering:** The software requirements specification (SRS) encompasses the process of collecting and documenting the set of requirements that the software needs to adhere to. It may include a set of use cases that describe the business needs and user flows that the software must implement. Software requirements can be classified into four broad categories: functional, external and User Interface (UI), system features, and non-functional.
2. **Design:** Software design is the process of transforming the requirements into a structure that is implementable using code. The technical lead breaks down requirements into sets of related components with clearly defined behaviors, boundaries, and interactions. These components define the system architecture.
3. **Coding for Quality:** Code quality refers to the characteristics of the code, including attributes such as maintainability, readability, testability, and security. Coding for quality entails following a set of coding practices during development, such as following common coding standards, using automated tools known as linters, and commenting in the code itself.
4. **Testing:** Software testing is the process of verifying that the software matches established requirements and is free of bugs. Levels of testing include unit,

integration, system, and user acceptance testing (UAT). Testing can broadly be divided into three categories: functional, non-functional, and regression.

5. **Releases:** When the newest version of the software is distributed, it is referred to as a "release." Different types of releases are intended for different audiences, including alpha, beta, and general availability (GA) releases.
6. **Documenting:** Software documentation should be provided to both non-technical end-users and technical users. System documentation is geared towards the technical user, while user documentation is provided to the non-technical end-users in the form of user guides, instructional videos and manuals, online help, and inline help.

In this lecture, you learned that:

- Requirement gathering is collecting and documenting the set of requirements that the software needs to adhere to.
- Designing transforms requirements into a structure that developers can use.
- Coding for quality entails following a set of coding practices during development.
- Testing is the process of verifying that the software matches established requirements and is free of bugs.
- There are three types of releases: alpha, beta, and general availability.
- Documenting requires text or video that explains the software to technical and non-technical users.

Requirements

After this lecture, you will be able to:

- Describe the steps of the requirement gathering process.
- Explain the purpose of a User Requirement Specification (URS) document.
- Explain the purpose of a Software Requirement Specification (SRS) document.
- Explain the purpose of a System Requirement Specification (SysRS) document.

Requirement Gathering Process: The requirement gathering process is a six-step process that includes:

1. **Identifying Stakeholders:** Involves identifying key personnel from various groups affected by the product, such as decision-makers, end-users, and support personnel.
2. **Establishing Goals and Objectives:** Defining broad, long-term outcomes (goals) and more specific, actionable, and measurable actions (objectives).

3. **Eliciting Requirements:** Gathering requirements through surveys, questionnaires, and interviews, and ensuring they align with the goals and objectives.
4. **Documenting Requirements:** Ensuring requirements are documented clearly and understood by stakeholders and the project team.
5. **Analyzing and Confirming Requirements:** Analyzing requirements for consistency, clarity, and completeness, and getting approval from stakeholders.
6. **Prioritizing:** Labeling requirements as "must-have," "highly desired," or "nice to have," and ordering them within these categories.

User Requirement Specification (URS): The URS document describes the business needs and expectations of the end-users from the software system. It is written as user stories or use cases that answer who the user is, what function needs to be performed, and why the user wants this functionality. User acceptance testing determines if these requirements have been met.

Software Requirement Specification (SRS): The SRS document captures the functionalities that the software should perform and establishes benchmarks for its performance. It includes a purpose statement, scope, constraints, assumptions, dependencies, and requirements sorted into four categories: functional requirements, external interface requirements, system features, and non-functional requirements.

System Requirement Specification (SysRS): The SysRS document outlines the requirements of an entire system, including system capabilities, interfaces, user characteristics, policy requirements, regulation requirements, personnel requirements, performance requirements, security requirements, system acceptance criteria, and hardware requirements.

Conclusion: The requirement gathering process is crucial for defining and documenting the problem to be solved and how to solve it. The URS, SRS, and SysRS documents play a vital role in capturing and communicating the expectations and requirements of the software system to ensure its success.

Software Development Methodologies

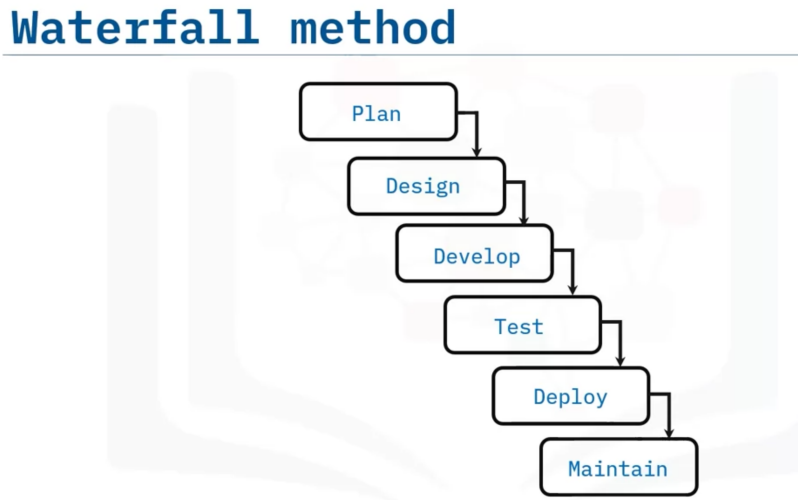
In this lecture, you will learn about different approaches to the software development life cycle (SDLC), with a focus on the Waterfall, V-Shape, and Agile methods.

Common Approaches to SDLC:

- **Waterfall:** A sequential method where each phase must be completed before the next begins. It is characterized by upfront planning and long intervals between releases.
- **V-Shape:** Similar to Waterfall, but with a focus on corresponding verification and validation phases, forming a V shape. It emphasizes testing at each stage of development.
- **Agile:** An iterative approach that emphasizes collaboration, customer feedback, and rapid, flexible response to change. It involves short cycles or sprints.

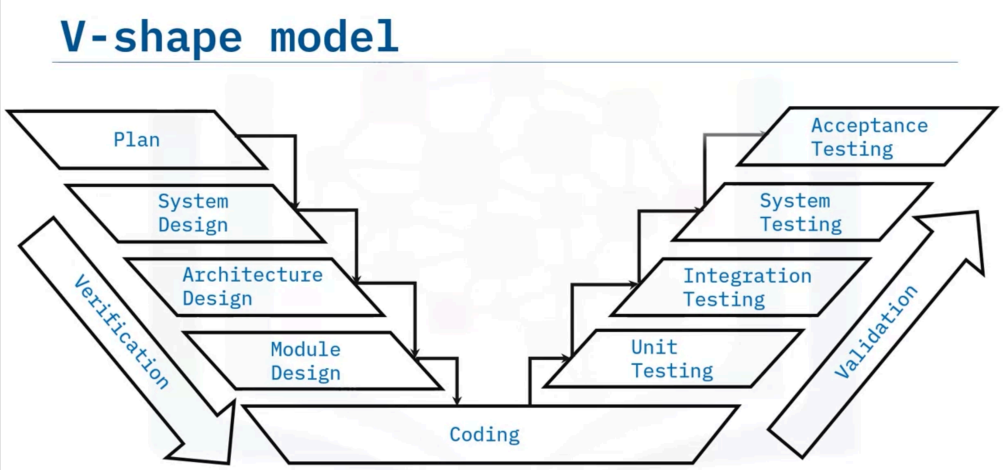
Waterfall Method:

- **Pros:** Easy to understand and follow, with discrete, well-defined stages. Upfront planning makes budgeting and resource allocation easier.
- **Cons:** Lacks flexibility to accommodate changes. Any alterations in requirements can be difficult to incorporate later in the process.



V-Shape Model:

- **Pros:** Simple and easy to use, with a clear emphasis on testing during the verification phase, saving time during coding and validation.
- **Cons:** Similar to Waterfall, it does not easily accommodate changing requirements. Alterations late in the process can be challenging.



Agile Method:

- **Pros:** Adapts well to changing requirements with ongoing research, planning, and testing. Encourages stakeholder and customer feedback, providing working code at the end of each sprint.
- **Cons:** Upfront planning, such as budgeting and scheduling, can be challenging due to the less defined scope of the product.