What Is Software Engineering?

After this lecture, you will be able to:

- Define software engineering
- List the responsibilities of a software engineer
- Compare and contrast software developers and software engineers

Definition of Software Engineering: Software engineering is the application of scientific principles to the design and creation of software. It employs a systematic approach to collect and analyze business **requirements**, with the goal of **designing**, **building**, and **testing** software applications to fulfill those requirements.

Evolution of Software Engineering:

- Initially undefined, software engineering has evolved into a modernized field since computing began in the late 1950s.
- Became a distinct discipline in the 1960s, evolving with new technologies and more scientific approaches.
- Shifted from ad-hoc programming to more formalized and standardized methods, addressing the challenges of the "Software Crisis."

The Software Crisis:

- Spanned from the mid-1960s to the mid-1980s, characterized by projects running over budget, behind schedule, and producing buggy code.
- Led to the realization that ad-hoc coding efforts needed to transform into an organized engineering discipline.
- Solutions included the adoption of standardized methodologies and the rise of computer-aided software engineering (CASE) tools.

CASE Tools: CASE tools facilitate software development in six key areas:

1. Business analysis and modeling
2. Development tools, including debugging environments
3. Verification and validation
4. Configuration management
5. Metrics and measurement
6. Project management

Software Engineers vs. Software Developers:

- Software engineers have a broader knowledge base and take a systematic, big-picture approach to software development.
- Software developers focus on writing code to implement specific functionality within a system.
- Engineers are often involved in larger scale projects, focusing on the structure and maintenance of software systems.

Responsibilities of a Software Engineer:

- Designing, building, and maintaining software systems
- Writing and testing code
- Consulting with stakeholders, third-party vendors, security specialists, and team members

Software Development Lifecycle (SDLC): The SDLC guides the development process to produce high-quality software, identifying the necessary steps for effective development.

Key Takeaways:

- Software engineering involves a systematic approach to software design and development.
- Software engineers have responsibilities that include system design and maintenance, code writing, and stakeholder consultation.
- The distinction between software engineers and developers lies in the scale and scope of their work, with engineers focusing on system-wide solutions and developers on specific functionalities.

Introduction to the Software Development Life Cycle (SDLC)

After this lecture, you will be able to:

- Describe what the Software Development Life Cycle (SDLC) is
- Explain the history of the SDLC
- Discuss some key advantages of using the SDLC

What is the SDLC? The SDLC is a systematic process used to develop high-quality software within a predictable timeframe and budget. Its goal is to produce software that meets a client's business requirements, with distinct phases that encompass their own processes and deliverables. The SDLC follows a cycle of planning, design, and development, which can be implemented as an iterative approach to software development.

History of the SDLC:

- The SDLC emerged in the mid-1960s as software development grew in complexity.
- Initially, it adopted the "waterfall method," a linear approach with discrete stages.
- Over time, the SDLC has adapted to more iterative methods to address changing customer needs and requirements.

Key Advantages of the SDLC:

1. **Provides a Structured Process:** The SDLC offers a roadmap for development teams, improving efficiency and reducing risks compared to ad-hoc approaches.
2. **Well-defined Phases:** Each phase of the SDLC is clearly defined, helping team members understand their tasks and timelines.
3. **Facilitates Communication:** The structured phases enhance communication between the customer, stakeholders, and the development team, providing clarity on the overall process.
4. **Supports Iteration:** The SDLC allows for iteration, enabling the incorporation of additional requirements at the end of a cycle.
5. **Early Problem Solving:** Issues are addressed early in the cycle, particularly during the design phase, minimizing problems during coding.
6. **Clear Roles and Responsibilities:** Each team member has a well-defined role, reducing conflict and overlapping responsibilities.

Conclusion: The SDLC provides a systematic approach to software development, addressing the challenges of complexity and changing requirements. Its structured

process, clear phases, and emphasis on communication and iteration offer significant advantages for efficient and effective software development.

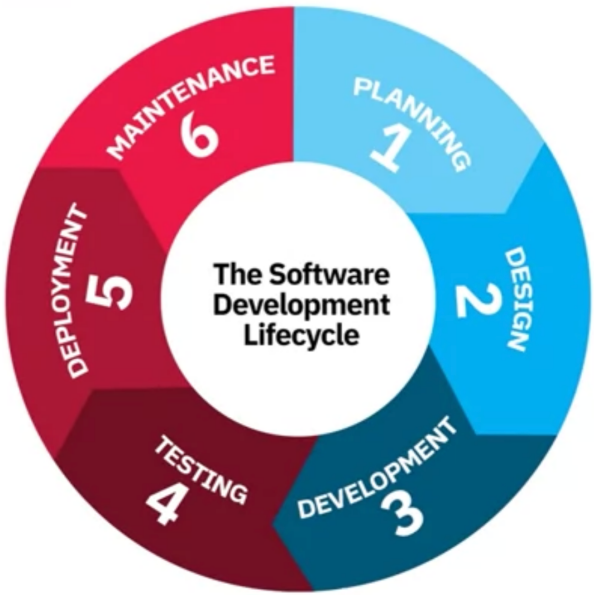
Phases of the Software Development Life Cycle (SDLC)

After this lecture, you will be able to:

- Name the phases involved in the SDLC
- Describe each phase
- Identify several tasks associated with each phase

Phases of the SDLC: The SDLC consists of six distinct phases:

1. **Planning**
2. **Design**
3. **Development**
4. **Testing**
5. **Deployment**
6. **Maintenance**



Each phase is discrete, meaning tasks from one phase do not overlap with those in the next. The SDLC was initially based on the traditional waterfall method but has since adapted to include iteration for accommodating shifting requirements.

Planning Phase:

- Gather, analyze, document, and prioritize requirements
- Consider factors such as users, purpose, data inputs and outputs, compliance, risk, quality assurance, resource allocation, and scheduling
- Estimate costs and time constraints, identify project teams, and define roles
- Use prototypes to clarify requirements if necessary
- Produce a Software Requirements Specification (SRS) document for approval by stakeholders

Design Phase:

- Develop software architecture based on requirements from the SRS
- Collaborate with team members to design the architecture
- Review architecture with stakeholders and team
- Design prototypes for demonstration purposes
- Create a design document for use in the development phase

Development Phase:

- Begin coding based on the design document
- Assign coding tasks using project planners
- Utilize programming tools, languages, and software stacks
- Adhere to organizational standards or guidelines

Testing Phase:

- Test code for stability, security, and compliance with SRS requirements
- Employ manual, automated, or hybrid testing methods
- Report, track, and fix bugs, retesting code until stable
- Conduct unit, integration, system, and acceptance testing

Deployment Phase:

- Release the application into the production environment
- Deploy in stages, starting with a user acceptance testing (UAT) platform
- Release to production upon customer approval
- Make software available on various platforms, such as websites, app stores, or corporate networks