SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ECT312 – EMBEDDED SYSTEM DESIGN

III YEAR/ VI SEMESTER

1

UNIT 5 : SYSTEM DESIGN TECHNIQUES AND REAL TIME CONCEPTS

TOPIC 1 & 2 : Design Methodologies & System Analysis and Architecture Design

Design Methodologies - Introduction

- ❖ Embedded systems are the backbone of modern technology, powering devices and machinery across various industries

- ❖ design of embedded systems presents unique challenges due to constraints such as limited resources, real-time requirements, and diverse application domains

- ❖ design methodologies play a crucial role in addressing these challenges and ensuring the efficiency and reliability of embedded systems

Design Methodologies

Importance of Design Methodologies

- Design methodologies provide structured approaches and frameworks for managing the complexity of embedded systems design

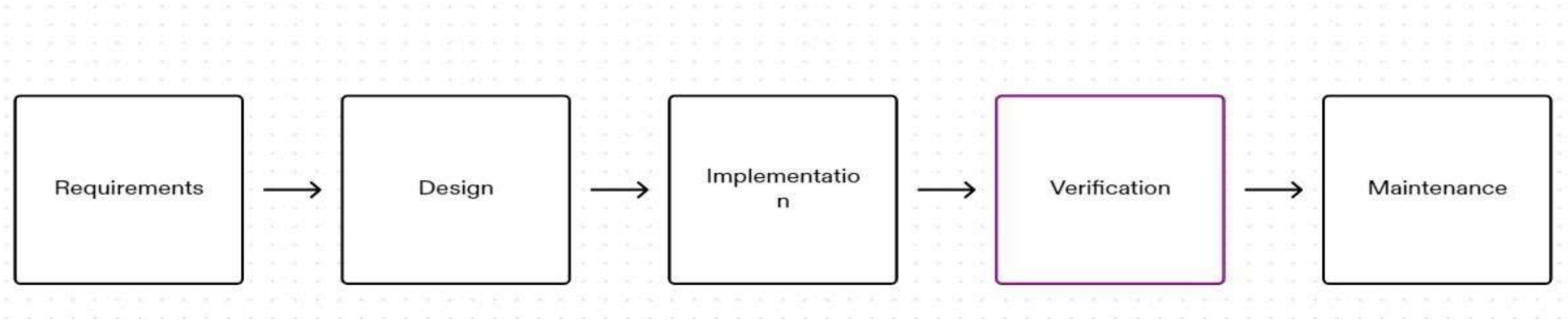
- They offer systematic methods for requirement analysis, system architecture design, implementation, testing, and maintenance

- By following established design methodologies, embedded systems designers can streamline the development process, reduce risks, and improve the quality of the final product.

Design Methodologies

WATERFALL MODEL

- ❖ Overview of the Waterfall Model and its application in embedded systems design.
- ❖ Phases: Requirements, Design, Implementation, Testing, Deployment, Maintenance.
- ❖ Advantages: Clear structure, well-defined milestones, and documentation.
- ❖ Disadvantages: Limited flexibility, challenges in accommodating changes late in the process.



Design Methodologies

AGILE METHODOLOGY

- ❖ Agile development is an iterative and flexible approach to design that emphasizes collaboration, adaptability, and continuous improvement. It involves breaking down the development process into small incremental steps, allowing for frequent feedback and adjustments. The key principles of agile development include:
  - ➢ Iterative development: breaking down the project into small iterations or sprints, with each iteration delivering a working product.
  - ➢ Customer collaboration: involving the customer throughout the development process to ensure their needs are met.
  - ➢ Continuous improvement: regularly reviewing and refining the design based on feedback and lessons learned.
  - ➢ Flexibility: being able to adapt to changing requirements and priorities.
- ❖ Agile development methodologies, such as Scrum and Kanban, provide frameworks and tools to implement these principles effectively.

Design Methodologies

ITERATIVE DEVELOPMENT

❖ Iterative development in embedded systems means building software step by step, fixing and improving it along the way.

➢ **Little by Little**: Instead of doing everything at once, developers work on small pieces of the software at a time. Each piece they finish works, even if it's not complete.

➢ **Feedback Helps**: They show what they've done to others early on, so they can get feedback. This helps them know if they're on the right track.

➢ **Adapting to Change**: Embedded systems often need to change because of new requirements or problems. Iterative development lets developers adjust the software as they go, making it easier to handle changes.

Design Methodologies

SPIRAL MODEL

The Spiral Model is like a roadmap for building embedded systems, which are devices with both hardware and software, like smartphones or smartwatches.

1.Setting Goals: First, we figure out what we want our embedded system to do and what limitations we have to work with, like how much memory or battery power it can use.

2.Thinking About Risks: Instead of just diving in, we spend time thinking about what could go wrong. This could be problems with the hardware, software, or how they work together.

3.Making Test Versions: We build simple versions of our system to test out ideas and see if they work. These aren't the final product; they're just to help us learn and fix problems early.

4.Building a Bit at a Time: We don't try to make everything all at once. Instead, we add small pieces of the system step by step.

Design Methodologies

DESIGN SPRINTS

Design sprints are a way to solve problems and test ideas fast.

1.Get Ready: Gather a team with different skills, like engineers and designers. Figure out what problem you're trying to solve.

2.Learn About the System: Understand the embedded system you're working on, including what users need and what the technical limits are.

3.Brainstorm Ideas: Everyone comes up with ideas for solving the problem, thinking about both the hardware and software parts of the system.

4.Pick the Best Ideas: Share and discuss the ideas. Choose the ones that seem the most promising and realistic.

Design Methodologies

SUCCESSIVE REFINEMENT

Successive refinement is a design methodology that involves breaking down a complex problem or system into smaller, more manageable parts, and then refining each part successively until it becomes detailed enough to be implemented.

1.**Start with the Big Picture**: First, you look at the whole puzzle and understand what it's supposed to look like in the end.

2.**Break it Down**: Then, you start breaking the puzzle into smaller sections or pieces. Each piece represents a part of the overall picture.

3.**Work on Each Piece**: You focus on one piece at a time, adding more details and making sure it fits perfectly with the neighboring pieces.

4.**Iterate and Improve**: As you work on each piece, you might go back and forth, making adjustments until it looks just right. You might even ask for feedback from others to make sure you're on the right track.

Design Methodologies

HIERARCHICAL DESIGN FLOWS

Hierarchical design flows are a structured approach to designing complex systems or products by organizing the design process into multiple hierarchical levels or layers. This methodology is commonly used in various design disciplines, including electronics, software engineering, and architecture.

1. **Starting at the Top**: You begin with the big picture in mind, like knowing you want to build a tall tower. This is called the top-down approach.
2. **Breaking it Down**: You break the tower into smaller sections or layers. Each layer represents a different part of the tower.
3. **Working on Each Layer**: You focus on one layer at a time, adding more details and making sure it fits with the layer below it.
4. **Setting Clear Boundaries**: You establish clear boundaries between each layer so they can interact properly. This helps each layer do its job without getting mixed up with the others.

Design Methodologies

TEST-DRIVEN DEVELOPMENT

Test-Driven Development (TDD) is a design methodology commonly used in embedded systems development to ensure software quality, reliability, and adherence to requirements.

1.Write Tests First: In TDD, developers start by writing automated tests for the functionality they want to implement.

2.Test Implementation: Once the tests are written, developers implement the functionality to make the tests pass.

3.Run Tests: After writing the initial code, developers run the automated tests to check if they pass. If any tests fail, developers iterate on the implementation until all tests pass successfully.

4.Refactor Code: Once the tests pass, developers may refactor the code to improve its structure, readability, and performance. Refactoring ensures that the code remains maintainable and scalable over time.

Design Methodologies

Hardware/Software Co-design

Hardware/Software Co-design is a design methodology commonly used in embedded systems development to optimize the interaction between hardware and software components.

1.Working Together: Hardware/Software Co-design means the hardware and software teams collaborate from the start. They don't work in isolation; they work together, like a team building a car engine and its control system simultaneously.

2.Understanding Needs: The process begins with understanding what the car needs to do. This includes how fast it should go, how much fuel it should use, and how it should respond to the driver's commands.

3.Talking to Each Other: Imagine the engine needs to tell the control system how fast it's spinning. They need to agree on how to communicate. This is like establishing interfaces between hardware and software components.

System Analysis

System analysis involves studying a system to understand its components, their interactions, and the system's behavior as a whole. This process aims to identify the requirements, constraints, and objectives of the system.

- ❖ Requirement gathering: Collecting and documenting the functional and non-functional requirements of the system from stakeholders.
- ❖ Feasibility study: Assessing the technical, economic, and operational feasibility of the proposed system.
- ❖ Modeling: Creating models such as use case diagrams, data flow diagrams, and entity-relationship diagrams to represent the system's structure and behavior.
- ❖ Risk analysis: Identifying potential risks and uncertainties associated with the system's development and operation.

System Analysis Process

Architecture design

Architecture design involves defining the structure, components, and interactions of a software system to meet its requirements while satisfying constraints such as performance, scalability, and maintainability. This phase translates the requirements identified during system analysis into a blueprint for the system's implementation.

- ❖ **Designing architectural patterns:** Choosing suitable architectural patterns such as client-server, layered, or microservices architecture based on the system's requirements.
- ❖ **Component design:** Identifying the major components of the system and defining their responsibilities and interfaces.
- ❖ **Allocation of functionality:** Assigning system functionalities to specific components and modules.

System Analysis Process

- **Identify Stakeholders**:
  - Identify all the stakeholders who will be involin or affected by the system. This includes end-users, administrators, managers, and any other relevant parties.
- **Define System Scope**:
  - Clearly define the boundaries and objectives of the system. Determine what the system will and will not do, and establish the goals it needs to achieve.
- **Gather Requirements**:
  - Gather requirements by interacting with stakeholders through interviews, surveys, workshops, or other techniques.
  - Identify functional requirements (what the system should do) and non-functional requirements (qualities the system should have, such as performance, security, usability, etc.).

System Analysis Process

❖ **Analyze Requirements**:

- ➢ Analyze and prioritize the gathered requirements to ensure they are complete, consistent, and feasible.
- ➢ Use techniques like requirement elicitation, validation, and negotiation to refine and clarify the requirements.

❖ **Model the System**:

- ➢ Create models to represent different aspects of the system, such as use case diagrams, data flow diagrams, entity-relationship diagrams, or process models.
- ➢ Models help visualize the system's structure, behavior, and interactions, aiding in understanding and communication.

❖ **Define System Interfaces**:

- ➢ Identify the interfaces between the system and its users, as well as any external systems or devices it interacts with.

Importance of System Analysis

1. **Risk Mitigation**: System analysis involves identifying and analyzing potential risks associated with the system's development and operation. By understanding these risks early on, developers can implement appropriate mitigation strategies to minimize the impact of potential problems on the project's success.
2. **Enhanced Communication**: System analysis facilitates effective communication between stakeholders, including end-users, clients, developers, and project managers. Clear documentation of requirements and system models helps in conveying ideas, clarifying expectations, and ensuring everyone is on the same page throughout the development process.
3. **Improved System Design**: A thorough system analysis lays the foundation for designing an efficient and robust system architecture. By understanding the system's structure, behavior, and interactions, architects and designers can make informed decisions about the system's design, leading to better performance, scalability, and maintainability.

Importance of System Analysis

4.Understanding User Needs: System analysis helps in understanding the needs, expectations, and requirements of users and stakeholders. By thoroughly analyzing the system's context and stakeholders' perspectives, developers can ensure that the final product aligns with users' expectations and fulfills their requirements.

5.Requirement Clarification: System analysis enables clear and detailed documentation of requirements. This helps in reducing ambiguity and misunderstandings between stakeholders and developers, leading to more accurate and effective system development.

6.Cost and Time Optimization: Identifying requirements accurately during the system analysis phase helps in preventing costly changes and rework later in the development process. It saves time and resources by addressing potential issues upfront, thereby optimizing the development process.

Types of Architecture Design

1. **Monolithic Architecture**:
   - In a monolithic architecture, the entire system is designed as a single, self-contained unit. All components and functionalities are tightly coupled and deployed together.
   - Monolithic architectures are straightforward to develop and deploy but may lack scalability and flexibility, especially in large or complex systems.

1. **Layered Architecture**:
   - Layered architecture organizes system components into layers, where each layer provides services to the layer above it and consumes services from the layer below it.
   - This architectural style promotes modularity, separation of concerns, and ease of maintenance. It is commonly used in embedded systems where clear separation of functionalities is desired.

Types of Architecture Design

3.Client-Server Architecture:

- Client-server architecture divides the system into client and server components, where clients request services or resources from servers.
- This architecture facilitates distributed computing, scalability, and separation of concerns. It is often used in networked embedded systems where devices interact with centralized servers or cloud services.

4.Event-Driven Architecture:

- Event-driven architecture decouples system components by using asynchronous communication based on events and event handlers.
- Components communicate by generating and consuming events, allowing for loose coupling, scalability, and responsiveness. This architecture is suitable for real-time embedded systems and systems with complex event processing requirements.

Types of Architecture Design

5.Service-Oriented Architecture (SOA):

- Service-oriented architecture decomposes the system into loosely coupled, reusable services that communicate via standardized interfaces.
- SOA promotes interoperability, flexibility, and reusability. It is often used in distributed embedded systems where interoperability between heterogeneous devices is essential.

6.Microservices Architecture:

- Microservices architecture decomposes the system into small, independently deployable services, each responsible for a specific functionality or domain.
- Microservices promote scalability, maintainability, and autonomy of development teams. They are commonly used in IoT ecosystems and cloud-connected embedded systems.

Architecture Design Process

1.Understand Requirements:

- Gather and analyze requirements from stakeholders to understand their needs, expectations, and constraints. Identify functional and non-functional requirements that will influence the system architecture.

2.Identify Stakeholders:

- Identify all stakeholders who will be involved in or affected by the system. This includes end-users, customers, project managers, developers, testers, and operations personnel.

3.Define System Scope:

- Define the boundaries and objectives of the system. Determine what the system will and will not do, and establish the goals it needs to achieve. Clarify the context in which the system will operate.

Architecture Design Process

4.Identify Architectural Drivers:

- ➢ Identify and prioritize architectural drivers such as performance, scalability, reliability, security, and maintainability. These drivers will guide architectural decisions throughout the design process.

5.Select Architectural Styles and Patterns:

- ➢ Choose appropriate architectural styles, patterns, and paradigms based on the identified requirements and architectural drivers. Consider factors such as system complexity, scalability, and development team expertise.

6.Decompose System:

- ➢ Decompose the system into smaller, manageable components or modules. Identify the responsibilities, interfaces, and dependencies of each component.

Importance Architecture Design

1.Blueprint for Development: Architecture design serves as a blueprint or roadmap for the development team. It provides a high-level view of the system's structure, behavior, and interactions, guiding developers in implementing the system's components and functionalities.

2.Scalability: A well-designed architecture enables the system to scale efficiently to accommodate changes in user requirements, data volume, and user load. It allows for the addition or modification of components without requiring extensive redesign or redevelopment.

3.Maintainability: A well-structured architecture promotes modularization and encapsulation, making it easier to maintain and update the system over time. It allows developers to make changes to individual components without impacting the entire system, reducing the risk of introducing bugs or errors.

Importance Architecture Design

4.Flexibility and Adaptability: An architecture that is designed with flexibility and adaptability in mind can easily accommodate changes in technology, business requirements, and market conditions. It allows the system to evolve over time to meet new challenges and opportunities.

5.Performance Optimization: Architecture design plays a crucial role in optimizing the system's performance. By carefully planning the distribution of responsibilities, data flow, and interactions between components, architects can ensure that the system meets performance requirements such as response time, throughput, and resource utilization.

6.Security: Architecture design influences the security posture of the system. By incorporating security principles and best practices into the architecture, such as authentication, authorization, encryption, and data protection mechanisms, architects can mitigate security risks and vulnerabilities.

Challenges In Architecture Design

1. **Complexity Management**: Handling the increasing complexity of software systems.
2. **Scalability**: Designing for growth without sacrificing performance or reliability.
3. **Performance Optimization**: Meeting performance requirements while minimizing bottlenecks.
4. **Maintainability and Evolvability**: Ensuring the architecture is easy to maintain and adapt over time.
5. **Security and Compliance**: Incorporating security measures and adhering to regulations.
6. **Interoperability and Integration**: Facilitating seamless integration with external systems.
7. **Technology Selection and Adoption**: Choosing the right tools and frameworks for the architecture.

Solutions In Architecture Design

1. **Modular Design**: Break down the system into modular components to manage complexity and promote maintainability.
2. **Scalability Planning**: Design for scalability by leveraging scalable technologies and architectures like microservices or distributed systems.
3. **Performance Profiling**: Use performance profiling tools and techniques to identify bottlenecks and optimize critical components.
4. **Continuous Refactoring**: Embrace continuous refactoring to keep the architecture adaptable and maintainable as requirements evolve.
5. **Security by Design**: Integrate security measures into the architecture from the outset, including encryption, access controls, and secure communication protocols.
6. **API and Interface Standards**: Define clear and standardized APIs and interfaces to facilitate interoperability and integration with external systems.
7. **Technology Evaluation**: Conduct thorough evaluations of technologies and

THANK YOU