**Module 5: Integration of Front-end and Back-end Systems**

**Frontend Development**

**1. Definition:**

- The part of the website users can see and interact with, including GUI, command line, design, navigation menus, texts, images, and videos.

**2. Essential Features:**

- User Interface Design

- Responsive Design

- HTML Structure

- Accessibility

- Integration

**3. Technologies and Tools:**

- Languages: HTML, CSS, JavaScript

- Frameworks/Libraries: AngularJs, React, jQuery, SASS, Flutter

- SEO, Usability, and Accessibility Testing

- Graphic Design and Image Editing Tools

- Web Performance and Browser Compatibility

**4. Web Accessibility:**

- Ensures all site visitors, regardless of ability, have a smooth experience.

- Allows people with disabilities equal access to digital platforms.

**5. Roles and Responsibilities of Frontend Developers:**

- Creating user-friendly and visually appealing interfaces.

- Writing clean HTML code and applying CSS styles.

- Implementing interactive elements using JavaScript.

- Developing layouts for different screen sizes.

- Testing and optimizing code for performance across browsers.

- Applying web accessibility best practices.

- Working with designers, backend developers, and stakeholders.

- Optimizing elements for faster page loading.

- Identifying and fixing frontend bugs.

**Backend Development**

1. Definition:

- The technical foundation of a software application or website that operates behind the scenes.

- Involves developing and maintaining server-side components.

**2. Essential Features:**

- Database Management

- Server Management

- Authentication and Authorization

- Data Processing

- API (Application Programming Interface)

**3. Technologies and Tools:**

- Languages: PHP, C++, Java, Python, Node.js

- Frameworks: Express, Django, Ruby on Rails, Laravel, Spring

**4. Roles and Responsibilities of Backend Developers:**

- Designing and maintaining databases.

- Optimizing queries.

- Designing and implementing APIs.

- Developing server-side logic.

- Implementing user authentication and authorization.

- Optimizing backend code.

- Implementing caching and data compression.

- Designing scalable systems.

- Managing server environments and configurations.

- Monitoring server health.

- Working with frontend developers to integrate components.

- Using version control systems like Git.

**Module 6: Software Testing**

**General Concepts**

1. Purpose of Testing:

- To show that a program does what it is intended to do and to discover defects.

- Involves executing a program with artificial data and checking for errors, anomalies, or non-functional attributes.

**2. Goals of Testing:**

- Validation Testing: Demonstrating that the software meets its requirements.

- Defect Testing: Discovering situations where the software behaves incorrectly.

**3. Verification vs. Validation:**

- Verification: "Are we building the product right?" - Ensuring the software conforms to its specification.

- Validation: "Are we building the right product?" - Ensuring the software meets user requirements.

**Testing Processes**

**1. Software Inspections:**

- Static verification technique involving analysis of the system representation to find problems.

- Can be applied before system implementation and does not require system execution.

- Effective in discovering program errors without the need for interaction between errors.

**2. Development Testing:**

- Unit Testing: Testing individual program units or object classes.

- Component Testing: Testing the interaction of multiple units.

- System Testing: Testing the complete integrated system.

**3. Automated Testing:**

- Utilizing frameworks like JUnit to write and run tests without manual intervention.

- Components include setup, call, and assertion parts.

**4. Testing Strategies:**

- Partition Testing: Identifying input groups with common characteristics.

- Guideline-based Testing: Using testing guidelines based on common errors made by developers.

**5. Types of Testing:**

- User Testing: Testing by users in their own environment.

- Release Testing: Testing a complete version of the system before release.

**6. Key Points:**

- Testing can only show the presence of errors, not their absence.

- Both inspections and testing should be used in the verification and validation process.

- Development testing is conducted by the development team, while a separate team handles system testing before release.

**Reviewer for Module 4: System Modeling**

1. Introduction to System Modeling

* Definition: Developing abstract models of a system to present different views or perspectives.
* Purpose: Helps analysts understand system functionality and communicate with customers.
* Key Concept: Graphical notation based on UML (Unified Modeling Language).

**2. Topics Covered**

* Context Models: Show operational context and boundaries of the system.
* Interaction Models: Describe interactions between system components or between the system and its environment.
* Structural Models: Represent the organization of a system and data structure.
* Behavioral Models: Illustrate the dynamic behavior of the system in response to events.
* Model-Driven Engineering: Automatically generating system implementations from models.

**3. Types of System Models**

* Existing System Models: Used during requirements engineering to understand current system operations.
* Planned System Models: Help in discussing design proposals and documenting new systems.

**4. Perspectives in System Modeling**

* External Perspective: Modeling the context or environment of the system.
* Interaction Perspective: Modeling interactions within the system or with the environment.
* Structural Perspective: Modeling the organization and structure of the system.
* Behavioral Perspective: Modeling the dynamic behavior of the system.

**5. UML Diagram Types**

* Activity Diagrams: Show the flow of activities in a process.
* Use Case Diagrams: Show interactions between a system and external actors.
* Sequence Diagrams: Show the sequence of interactions over time.
* Class Diagrams: Show the static structure of the system, including classes and associations.
* State Diagrams: Show the system’s response to internal and external events.

**6. Graphical Models Usage**

* Facilitating Discussion: Incomplete models can support discussions about proposed systems.
* Documentation: Accurate representation of existing systems.
* System Description: Correct and complete models can generate system implementations.

**7. Context Models**

* Definition: Illustrate the operational context and boundaries of a system.
* Importance: Helps in defining what is inside and outside the system.

**8. Process Models**

* Definition: Reveal how the system is used within broader business processes.
* Example: UML activity diagrams.

**9. Interaction Models**

* Purpose: Important for identifying user requirements and understanding system communication.
* Tools: Use case diagrams and sequence diagrams.

**10. Structural Models**

* Purpose: Display the organization of a system in terms of its components and their relationships.
* Tools: Class diagrams.

**11. Behavioral Models**

* Purpose: Represent the dynamic behavior of a system during execution.
* Types: Data-driven modeling and event-driven modeling.
* Tools: State machine models and statecharts.

**12. Model-Driven Engineering (MDE)**

* Approach: Models are the principal outputs, and code is generated from these models.
* Advantages: Higher levels of abstraction and easier adaptation to new platforms.
* Challenges: Developing translators for new platforms can be costly.

**13. Key Points to Remember**

* A model is an abstract view of a system.
* Different models provide complementary perspectives.
* Context, interaction, and structural models are fundamental for comprehensive system understanding.
* UML diagrams are essential tools for creating these models.