Course: Computer science Projects

Tutor: Dr. Sharon Yalov Handzel

Lesson 5: Functional and Technological Requirements

Objectives:

\* How to define and describe the functional requirements

\* Identifying technological requirements

**Functional Requirements**

- Identify the key functions and features the system needs to have from the end user's perspective. These should specify what the system should do.

- Document use cases to capture the interactions between users and the system. Include pre-conditions, post-conditions, basic flow, and alternate flows.

- Create user stories to document functional requirements from the user's perspective. Include acceptance criteria.

- Prioritize and estimate the effort for each requirement.

1. \*\*Identify Stakeholders:\*\*

- Determine who the stakeholders are. Stakeholders can include end-users, customers, project managers, developers, and any other individuals or groups affected by or influencing the project.

2. \*\*Conduct Stakeholder Interviews / workshops / Surveys

- Schedule interviews with key stakeholders to understand their perspectives, expectations, and requirements.

- Ask open-ended questions to encourage detailed responses and gather insights into the problem domain.

- Conduct workshops or focus group sessions involving relevant stakeholders to facilitate group discussions.

- Use collaborative techniques like brainstorming to elicit ideas and requirements.

- Distribute surveys or questionnaires to a wider audience to collect quantitative data on preferences or opinions.

- Analyze the survey results to identify common patterns or trends.

3. \*\*Review Existing Documentation:\*\*

- Examine any existing documentation, such as business plans, project charters, or relevant reports, to gain insights into the project's context.

4. \*\*Identify Functional and Non-Functional Requirements:\*\*

- Distinguish between functional requirements (features and capabilities) and non-functional requirements (performance, security, usability, etc.).

- Prioritize requirements based on their importance to stakeholders.

5. \*\*Use Case Analysis:\*\*

- Develop use cases to describe interactions between users and the system. This helps in understanding the system from an end-user perspective.

**Example of functional requirements:**

Consider a Task Management System which is a software application designed to help users organize, assign, and track tasks within a team or organization.

1. \*\*User Authentication and Authorization:\*\*

- Users should be able to register for an account and log in securely.

- Different user roles (e.g., Admin, Team Member) should have specific permissions and access levels.

2. \*\*Task Creation and Assignment:\*\*

- Users with the role of Admin should be able to create new tasks.

- Tasks should include a title, description, due date, priority level, and assignee.

- Users should be able to assign tasks to specific team members.

3. \*\*Task Tracking:\*\*

- Users should be able to view a dashboard displaying all tasks.

- The dashboard should show task status (e.g., Open, In Progress, Completed) and due dates.

- Users should have the ability to filter tasks based on different criteria (e.g., assignee, priority).

4. \*\*Task Editing and Updating:\*\*

- Users should be able to edit task details, including title, description, due date, and assignee.

- The system should log and display the history of changes made to a task.

5. \*\*Notifications:\*\*

- Users should receive notifications for upcoming task deadlines.

- Users should be notified when they are assigned a new task or when a task is updated.

6. \*\*Commenting and Communication:\*\*

- Users should be able to add comments to tasks for communication within the system.

- The system should support real-time updates or notifications for new comments.

7. \*\*File Attachments:\*\*

- Users should be able to attach files (documents, images) to tasks.

- The system should support file versioning and storage.

8. \*\*Search Functionality:\*\*

- Users should be able to search for tasks based on keywords, assignees, or other relevant criteria.

9. \*\*Reporting:\*\*

- Admin users should have access to reporting features, such as task completion rates, overdue tasks, and team performance metrics.

10. \*\*Integration with Calendar:\*\*

- The system should integrate with calendar applications to display task due dates alongside other appointments.

11. \*\*Mobile Accessibility:\*\*

- The system should be accessible via mobile devices, with a responsive design for different screen sizes.

12. \*\*Data Security:\*\*

- The system should implement security measures to protect user data and ensure confidentiality.

**Architectural Requirements**

- Specify the high-level components of the system and how they will interact. These may include databases, APIs, web interfaces, mobile apps, etc.

- Choose architectural patterns like MVC, client-server, n-tier, microservices, etc. based on needs.

- Document the key technologies to be used. Consider trade-offs like ease of development vs performance.

- Create sequence and communication diagrams to illustrate component interactions.

- Scale requirements should inform how components can be replicated or distributed.

1. \*\*System Architecture:\*\*

- Define the overall system architecture, including components, modules, and their interactions.

- Specify whether the system will follow a monolithic architecture, microservices, or any other architectural style.

2. \*\*Scalability:\*\*

- Define how the system should scale to accommodate an increase in users or data volume.

- Specify scalability requirements for both vertical (scaling up) and horizontal (scaling out) scenarios.

3. \*\*Performance:\*\*

- Set performance requirements such as response times, throughput, and resource utilization.

- Identify key performance indicators and acceptable performance thresholds.

4. \*\*Reliability and Availability:\*\*

- Define the system's reliability requirements, including expected uptime and mean time between failures.

- Specify any redundancy or failover mechanisms to ensure high availability.

5. \*\*Security:\*\*

- Outline security requirements, including authentication, authorization, data encryption, and secure communication protocols.

- Specify compliance with relevant security standards and regulations.

6. \*\*Data Architecture:\*\*

- Define the data architecture, including databases, data models, and data flow within the system.

- Specify requirements for data consistency, integrity, and availability.

7. \*\*Integration:\*\*

- Identify external systems or services that the software must integrate with.

- Specify the protocols, APIs, or communication standards for seamless integration.

8. \*\*Scalability:\*\*

- Define how the system should scale to accommodate an increase in users or data volume.

- Specify scalability requirements for both vertical (scaling up) and horizontal (scaling out) scenarios.

**Common architectures:**

1. \*\*Monolithic Architecture:\*\*

- \*\*Description:\*\* In a monolithic architecture, the entire application is built as a single, self-contained unit. All components, such as the user interface, business logic, and data access, are tightly integrated.

- \*\*Use Case:\*\* Suitable for small to medium-sized applications where simplicity and ease of deployment are priorities.

2. \*\*Microservices Architecture:\*\*

- \*\*Description:\*\* Microservices architecture decomposes an application into a set of small, independent services that communicate with each other through well-defined APIs. Each microservice focuses on a specific business capability.

- \*\*Use Case:\*\* Ideal for large and complex systems where scalability, maintainability, and independent deployment of services are crucial.

3. \*\*Service-Oriented Architecture (SOA):\*\*

- \*\*Description:\*\* SOA is an architectural style that involves designing software components as services. These services communicate through a network and can be combined to create larger applications.

- \*\*Use Case:\*\* Suitable for distributed systems, where interoperability and reusability of services are essential.

4. \*\*Client-Server Architecture:\*\*

- \*\*Description:\*\* In client-server architecture, the application is divided into two parts: a client (user interface) and a server (back-end). Clients request services or resources from servers, and servers fulfill those requests.

- \*\*Use Case:\*\* Commonly used in networked applications where a separation of concerns between the client and server is necessary.

5. \*\*Event-Driven Architecture (EDA):\*\*

- \*\*Description:\*\* In EDA, components communicate through events, and the system responds to events triggered by changes in state. This can lead to loosely coupled and highly scalable systems.

- \*\*Use Case:\*\* Suitable for systems that need to react to and process events in real-time, such as financial trading systems or IoT applications.

6. \*\*Layered Architecture:\*\*

- \*\*Description:\*\* In a layered architecture, the application is organized into layers, each responsible for a specific functionality. Common layers include presentation, business logic, and data storage.

- \*\*Use Case:\*\* Provides a modular structure, making it easy to understand and maintain. Often used in conjunction with other architectural patterns.

7. \*\*Repository Pattern:\*\*

- \*\*Description:\*\* The repository pattern separates the logic that retrieves data from the underlying storage. It provides a consistent interface for accessing data and helps decouple the application from the data storage implementation.

- \*\*Use Case:\*\* Particularly useful in applications where the choice of data storage (database, external API, etc.) may change.

8. \*\*MVC (Model-View-Controller):\*\*

- \*\*Description:\*\* MVC is a design pattern where the application is divided into three interconnected components: Model (data and business logic), View (user interface), and Controller (handles user input and updates the model).

- \*\*Use Case:\*\* Widely used in web applications to separate concerns and improve maintainability.

**Example of architectural Requirements for E-commerce Platform:**

1. \*\*System Architecture:\*\*

- The system shall follow a microservices architecture to enable independent development, deployment, and scalability of individual business components.

2. \*\*Scalability:\*\*

- The system shall be designed to scale horizontally to handle increased user traffic during peak periods, such as holiday seasons.

- The architecture should support dynamic scaling based on demand.

3. \*\*Performance:\*\*

- The platform shall provide a response time of no more than 2 seconds for critical user interactions, such as page loads and search queries.

- The system shall support a minimum of 10,000 concurrent users without significant degradation in performance.

4. \*\*Reliability and Availability:\*\*

- The system shall have an availability target of 99.9%, allowing for scheduled maintenance windows.

- Redundancy and failover mechanisms shall be implemented to minimize downtime in case of component failures.

5. \*\*Security:\*\*

- The platform shall adhere to industry-standard security practices, including data encryption in transit and at rest.

- User authentication and authorization mechanisms shall be implemented securely.

6. \*\*Data Architecture:\*\*

- The system shall utilize a distributed database architecture, ensuring data consistency and availability across multiple data centers.

- Data backups shall be performed regularly, and disaster recovery procedures shall be in place.

7. \*\*Integration:\*\*

- The platform shall integrate with external payment gateways, shipping providers, and inventory management systems using industry-standard APIs.

- Integration points should be designed with scalability and fault tolerance in mind.

8. \*\*Scalability:\*\*

- The system shall be capable of scaling to handle an increase in the number of products, categories, and concurrent user sessions.

- Load balancing mechanisms shall be implemented to distribute traffic evenly across servers.

9. \*\*User Experience:\*\*

- The architecture shall support responsive design to provide a consistent user experience across various devices and screen sizes.

- CDN (Content Delivery Network) integration shall be considered to optimize content delivery.

10. \*\*Monitoring and Logging:\*\*

- The platform shall have comprehensive monitoring capabilities, including real-time performance metrics, error tracking, and log aggregation.

- Automated alerts shall be configured for critical system events, ensuring timely responses to issues.

11. \*\*Compliance:\*\*

- The architecture shall comply with relevant data protection and privacy regulations, such as GDPR or HIPAA.

- Accessibility standards (e.g., WCAG) shall be considered for inclusive user experiences.

12. \*\*Technology Stack:\*\*

- The platform shall use modern and well-supported technologies, including a specific programming language, web framework, and database management system.

- Third-party libraries and frameworks should be carefully selected and regularly updated.

**Technological Requirements**

- List out key technical requirements like target browsers, devices, OS, frameworks, programming languages, etc.

- Specify standards like HTTP, REST, JSON, testing frameworks, etc.

- Identify security requirements for data, APIs, encryption, access control, etc.

- Define infrastructure requirements for hosting platforms, servers, databases, caching, etc.

- Consider how technologies will impact ease of development, hiring needs, scalability, and maintenance.

1. \*\*Programming Languages and Frameworks:\*\*

- Specify the programming languages and frameworks to be used for development.

- Consider factors such as developer expertise, community support, and scalability.

2. \*\*Database Technology:\*\*

- Define the database management system (DBMS) and any specific database technologies.

- Specify requirements for data storage, retrieval, and scalability.

3. \*\*Frontend Technologies:\*\*

- Specify the frontend technologies, such as web frameworks or mobile development frameworks.

- Consider factors like user experience, responsiveness, and cross-browser compatibility.

4. \*\*Backend Technologies:\*\*

- Identify the server-side technologies, including server frameworks, middleware, and application servers.

- Specify requirements for handling business logic, data processing, and communication with databases.

5. \*\*Cloud Services:\*\*

- If applicable, define the use of cloud services (e.g., AWS, Azure, Google Cloud) and specify requirements for deployment, scalability, and maintenance.

6. \*\*Development Tools:\*\*

- Specify the development tools, version control systems, and collaboration platforms to be used.

- Define coding standards, testing frameworks, and continuous integration/continuous deployment (CI/CD) practices.

7. \*\*Security Tools:\*\*

- Identify any security tools or frameworks to be used for code analysis, vulnerability scanning, and penetration testing.

8. \*\*Compliance and Standards:\*\*

- Specify any industry standards, compliance requirements, or regulations that the software must adhere to.

9. \*\*Documentation Standards:\*\*

- Define standards for code documentation, API documentation, and system architecture documentation.

10. \*\*Testing and Quality Assurance:\*\*

- Specify testing frameworks, automated testing tools, and quality assurance processes.

- Define performance testing tools and methodologies.