The University of Azad Jammu and Kashmir

Lab Task # 01

Course Instructor: Engr. Sidra Rafique

Semester: Fall-2024

Session: 2022-2026

Submission Date: Oct 21, 2024

Submitted By: Syeda Urwa Ajmal

Roll No: 2022-SE-16

Course Name: SC&D

Code: SE-3102

Basics of Software Construction and Development

Table of Contents

Basics of Software Construction and Development 1
1. Analysis 3
Problem Definition: 3
Requirement Gathering: 3
Analysis Phase: 3
Data Modeling: 3
Process Modeling: 3
Interface Modeling 4
Validation and Verification: 4
2. Components of Software model: 4
Use Cases: 4
Class Diagrams: 4
Sequence Diagrams: 4
State Machine Diagrams: 4
Activity Diagrams: 4
Data Flow Diagrams (DFDs): 5
Entity-Relationship (ER) Diagrams 5
User Stories: 5
3. Characteristics 5
Completeness: 5
Consistency: 5
Accuracy: 5
Relevance: 6
Unambiguity: 6
Flexibility: 6
Scalability: 6
Maintainability: 6
Difference between Flow-oriented and class-based modeling 7
1. Flow-Oriented Modeling: 7
2. Class-Based Modeling: 7
Example of University Management System (UMS) 8
- Analysis 8
- Applying Characteristics 9

1. Analysis

Problem Definition:
In the analysis phase, problem definition is the initial step where the scope and nature of the problem are clearly identified. This involves understanding the issues within the current system or situation, gathering input from stakeholders, and outlining specific requirements and objectives. By defining the problem accurately, teams ensure they address the right challenges and lay a foundation for effective solutions.

A well-defined problem statement includes:

- Symptoms and underlying causes of the issue.
- Impact on users, business, or system operations.
- Boundaries and constraints within which the solution must operate.

Requirement Gathering:

This is the initial phase where stakeholders identify and document system requirements. It involves interviews, surveys, and workshops to clarify needs and expectations for the software system.

Analysis Phase:

In this phase, requirements are analyzed to establish a clear understanding of system functionality. This involves identifying ambiguities, inconsistencies, and prioritizing requirements. A functional specification document is created to serve as a blueprint.

Data Modeling:

Data modeling defines the structure and relationships of data within the system. Common tools include ER (Entity-Relationship) diagrams that help visualize data entities and their relationships, which aids in database design.

Process Modeling: This involves mapping out workflows and interactions within the system, often through tools like DFDs (Data Flow Diagrams) and flowcharts. Process models show how data moves and is processed within the system.

Interface Modeling:

Interface modeling focuses on the user interaction aspects, defining how users will interact with the system. It includes wireframes, mockups, and prototypes to design an intuitive and user-friendly interface.

Validation and Verification: Validation ensures that the system meets stakeholder needs (building the right system), while verification checks if the system meets specifications (building the system right). This phase uses testing, reviews, and walkthroughs to catch issues before implementation.

2. Components of Software model:

Use Cases:
Use cases describe how a user interacts with the system to achieve a goal. They focus on user requirements and capture various scenarios to show how the system responds to user actions.

Class Diagrams:

These are part of UML (Unified Modeling Language) and depict the structure of a system by showing its classes, attributes, methods, and the relationships between classes. They are key for object-oriented modeling.

Sequence Diagrams:

Sequence diagrams represent the flow of messages or interactions between objects over time. They show the order of operations or events and are useful for detailing the sequence of steps in a process.

State Machine Diagrams:

These diagrams illustrate the different states an object can be in, as well as the transitions between those states in response to events. They are useful for modeling the behavior of an object over time.

Activity Diagrams:

Activity diagrams are flowchart-like diagrams that represent workflows or processes within the system. They show actions, decision points, and parallel activities, making them useful for modeling business processes or complex logic.

Data Flow Diagrams (DFDs):

DFDs show how data flows through a system and how it’s processed at different stages. They map out data sources, processes, data stores, and data destinations, helping to clarify data-related requirements.

Entity-Relationship (ER) Diagrams: ER diagrams model the data structure for a system by showing entities (data objects), attributes, and the relationships between them. They are essential for database design.

User Stories:

User stories are short, simple descriptions of a feature from the perspective of an end-user. They focus on what the user wants to accomplish, forming the foundation for requirements in agile methodologies.

3. Characteristics

Completeness:

Completeness ensures that all required features, functions, and data are defined in the system. Every necessary detail should be covered to avoid gaps that may cause functionality or performance issues.

Consistency:

Requirements and design must be free from contradictions and must align across the documentation. Consistency is essential to ensure that different parts of the system work harmoniously without conflicting requirements.

Accuracy:

Accurate requirements and design reflect the true needs and intentions of stakeholders. They prevent misunderstandings and reduce the risk of implementing features that do not meet user expectations.

Relevance:

Relevance ensures that only necessary and valuable requirements are included, avoiding unnecessary features that may lead to complexity or wasted resources.

Unambiguity:

Requirements and design should be clear and precise, leaving no room for multiple interpretations. Unambiguous specifications are essential for accurate development and testing.

Flexibility:

A flexible design can adapt to changes in requirements or the business environment. This adaptability helps extend the system’s lifespan and minimizes redesign efforts.

Scalability:

Scalability is the ability of the system to handle growth, whether in user volume, data size, or functionality. Scalable requirements and design allow the system to expand without major redesigns.

Maintainability:

Maintainable requirements and design make it easy to update, modify, or improve the system over time. They help ensure that the system remains functional and up-to-date with minimal effort.

Difference between Flow-oriented and class-based modeling

Flow-oriented and class-based modeling differ in focus and representation:

1. Flow-Oriented Modeling:

- Focuses on processes and data flow within the system.
- Uses Data Flow Diagrams (DFDs) to map how data moves through processes and subsystems.
- Emphasizes the sequence and interaction of data, suitable for understanding system behavior and information flow.

2. Class-Based Modeling:

- Centers on defining the structure and relationships of data and behaviors within a system.
- Uses Class Diagrams to represent classes, their attributes, methods, and relationships (e.g., inheritance, associations).
- Emphasizes data organization, making it ideal for object-oriented design and encapsulating system structure.

Example of University Management System (UMS)

Here's a concise example of a University Management System (UMS) that integrates each of the steps and characteristics discussed:

- **Analysis**

1. Requirement Gathering

- Stakeholders (e.g., university admin, professors, students) identify the needs: course registration, grading, attendance, fee management, and communication.

2. Analysis Phase

- Requirements are analyzed for clarity and feasibility. For example, "Students should be able to view grades online" is translated into functional specifications detailing access rights, update frequencies, and data visibility.

3. Data Modeling

- ER diagrams are used to represent entities such as **Student**, **Course**, **Professor**, and **Department**, with relationships (e.g., a student enrolls in multiple Courses, a professor teaches multiple Courses).

4. Process Modeling

- Data Flow Diagrams (DFDs) illustrate processes such as Student Registration, Course Enrollment, and Fee Payment, showing how data flows between them.

5. Interface Modeling

- Wireframes and mockups are designed for student and faculty portals, showing a user-friendly interface for course enrollment, grade viewing, and communication tools.

6. Validation and Verification

- Verification checks that each module (e.g., enrollment, grading) meets requirements and works as specified. Validation ensures the system supports user needs through user acceptance testing with a sample of students and faculty.

- **Applying Characteristics**

Completeness:

All necessary functions, such as registration, grade entry, and fee tracking, are defined.

Consistency:

Uniformity is maintained across modules so that similar actions, like logging in or viewing grades, work consistently.

Accuracy:

Stakeholder needs are accurately translated into system functions, such as the correct computation and display of grades.

Relevance:

Only essential features for university management are included, avoiding unnecessary complexity.

Unambiguity:

Requirements are specified clearly (e.g., "Students can view grades after faculty submission").

Flexibility:

Modular design allows the system to adapt, such as adding new course options without major changes.

Scalability:

The system supports more students and courses as the university grows.

Maintainability:

Clear documentation and modular code design make future updates and maintenance easy.

This UMS example illustrates how these phases and characteristics guide the development of a comprehensive, user-focused system.