**UML Design Modeling**

The University of Arizona Global Campus

CST499: Capstone for Computer Software Technology

Dr. Charmelia Butler

October 14th, 2024

The University Enrollment System, a crucial tool for managing academic registrations, requires robust design and testing to ensure its reliability and effectiveness. In today’s software development landscape, creating clear, well-documented system architecture is vital for managing complexity and ensuring smooth interaction between users and system components. This is where Unified Modeling Language (UML) diagrams come into play, helping developers visualize the system structure and behavior before the actual coding begins. Equally important is the rigorous testing process, which verifies that the system functions as intended and meets the needs of users. Comprehensive testing—spanning component, integration, system, and acceptance levels—helps identify and fix issues early in the software development life cycle. This paper will first explore six key UML diagrams that depict the architecture of a University Enrollment System using a client/server model. Following that, it will discuss the various levels of testing, examining how these processes are applied to ensure the system’s quality. Ultimately, this paper will demonstrate how effective design, coupled with thorough testing, leads to the successful deployment of a reliable, user-friendly enrollment system.

Before examining the UML diagrams, a brief explanation of the client/server architecture should be understood (Terra, 2024). The Client/Server Architecture is a network architecture where client devices (users) interact with server systems to perform tasks, request services, and access data. The client typically handles the user interface and presents information, while the server manages application logic, data storage, and processing.

In a University Enrollment System, the client could be any device that students, faculty, or staff use to interact with the system. These clients provide the user interface where students log in, search for courses, and register. The client sends HTTP requests to the web server and waits for the response. The user’s interaction occurs on the client side (viewing course lists, enrolling, etc.).

The server side is responsible for handling requests from the client and processing them. It includes the web server which manages HTTP requests from the client. In the enrollment system, the web server renders the pages or sends data based on requests, such as fetching course lists, handling login, and so forth. The SQL server stores all the university's data, including student records, courses, faculty, and enrollment information. When a student registers for a course, the enrollment information is updated in the database. As per the SRS, if the course is full, the student will be placed on a waiting list. The database server is responsible for handling queries, data retrieval, and updates.

In order to understand the structure and behavior of the University Enrollment System, six UML diagrams will be examined: Package Diagram, Deployment Diagram, Sequence Diagram, Class Diagram, State Diagram, and Use Case Diagram. Each of these diagrams provides a different perspective of the system's design, addressing key aspects of its functionality and architecture.

The Package Diagram as shown in figure 1 visually represents the organization of the system’s components into logical packages, such as the Client, Server, and Database (Visual Paradigm, 2023). It shows dependencies between these packages, making it easier to understand how they interact. For the enrollment system, the Client Package contains modules for login, course search, and registration. The Server Package manages application logic, while the Database Package stores student records and course information.

**Figure 1:**

Package diagram

A diagram of a package

Description automatically generated

The Deployment Diagram shown in figure 2 details how software components are distributed across hardware resources in the system (Visual Paradigm, 2023). For the University Enrollment System, this diagram includes a Client Node (representing the user's device), a Web Server Node (handling HTTP requests and running the application), and a Database Node (storing course and student data). Communication links between these nodes show how data flows between the client, server, and database during the course registration process.

**Figure 2:**

Deployment diagram

**A diagram of a computer

Description automatically generated**

A Sequence Diagram captures the interaction between different system components during a specific process—in this case, course registration as shown in figure 3 (Codebun, n.d.). The primary use is showing the interactions between a user and the system in the order in which they occur and are useful for documentation during the design phase of the developing systems expected behavior and aids in the transition from use case requirements to the next formal level of requirements refinement (Bell, 2023). Not only does this give guidance to the development team, it also aids the business staff in communicating how the system works.

It begins with a student logging in and progresses through searching for a course, sending requests to the server, and updating the database when the student registers for a course. This diagram provides a clear view of the step-by-step flow of actions between the User, Client, Web Server, and Database.

**Figure 3:**

Sequence diagram

**A diagram of a software program

Description automatically generated**

The Class Diagram models the system's static structure, representing its classes, attributes, methods, and relationships (Geeks for Geeks, 2024). In the enrollment system, classes include User, Student, Instructor, Course, and Enrollment as shown in figure 4. Relationships between these classes such as shown in figure 4 are crucial for understanding how data is managed—such as a student enrolling in a course or an instructor assigning grades.

(remainder of page intentionally blank for formatting)

**Figure 4:**

class diagram

**A diagram of a course

Description automatically generated**

The State Diagram captures the various states a Student undergoes when registering for a course. A State Diagram is useful to the development team, users, and documentation as a behavioral diagram to illustrate the condition of the system at a finite instance in time (Geeks for Geeks, 2024). The diagram uses state transitions to show actions required to move to a different state. As shown in figure 5, these states include logged In, search for courses, registering for course, registered, waitlisted, or dropped. Transitions between these states depend on specific user actions (e.g., logging in, searching for a course, registering) and system responses (e.g., course availability and confirmation on registration).

**Figure 5:**

State diagram

**A diagram of a course

Description automatically generated**

The System Use Case Diagram illustrates the interactions between system users (actors) and the system itself (Codebun, n.d.). Also known as behavior diagrams, they are used to describe a set of actions known as use cases that an actor could perform within the system. Every use case should show an observable and valuable result to the stakeholders of the system (Fakhroutdinov, 2024).

For the enrollment system, the key actors are Students, Instructors, and Admins, each interacting with different use cases like Register for Course, Assign Grades, and Manage Courses. This diagram in figure 6 emphasizes how users perform various tasks within the system and the functionality available to them.

**Figure 6:**

Use case diagram

**A diagram of a student

Description automatically generated**

Once the design of the University Enrollment System is mapped out using UML diagrams, the next step is to ensure the system operates as intended through a series of testing levels. Each level of testing focuses on a different aspect of the system, progressively verifying its components, interactions, overall functionality, and user satisfaction.

Component testing, also known as unit testing, focuses on individual parts or units of the system (Tsui, et al, 2016, 10.2.1) often corresponding to the classes defined in the Class Diagram. This type of testing verifies that each module, such as the Login Module or the Course Search Module, functions correctly in isolation. For example, tests would ensure that the login() method in the User class works as expected, verifying that users with valid credentials can log in and invalid credentials are rejected. The goal is to catch bugs early, ensuring that each component behaves as specified before integration.

After individual components have passed component testing, integration testing examines how these components work together (Tsui, et al, 2018, 10.2.1). This phase tests the interaction between the client, server, and database, as shown in the Sequence Diagram and Deployment Diagram. For example, integration testing would verify that after a student selects a course, the system correctly communicates with the database to check availability and update enrollment records. Issues like data mismatches or communication errors between components can be identified and addressed during this stage.

In system testing, the entire system is tested as a whole (Tusi, et al, 10.2.1) to ensure it meets the requirements specified in the Use Case Diagram. This phase simulates real-world scenarios where users log in, search for courses, register, and view their schedules. The system is tested for functionality, performance, security, and reliability. For instance, system testing would verify that students can successfully complete the entire registration process without errors and that the system can handle multiple users accessing it simultaneously.

Acceptance testing, also known as validation testing, is the final level of testing conducted to ensure that the system meets user expectations and business requirements (Tsui, et al, 2018, 10.2.1). This is often performed by actual end-users, such as students and administrators, in real-world conditions. Acceptance testing validates whether the system supports all use cases defined in the Use Case Diagram—for instance, checking that students can register for courses and instructors can assign grades. The successful completion of this phase is critical before the system goes live, as it confirms that the system is ready for deployment.

Designing and testing a complex system like the University Enrollment System requires careful planning and execution. The UML diagrams discussed in this paper provide a structured approach to visualizing the system’s architecture, interactions, and behaviors. From organizing components in a Package Diagram to detailing specific user interactions in a Use Case Diagram, these tools offer a comprehensive understanding of the system's functionality. Once the design is in place, testing at various levels—Component Testing, Integration Testing, System Testing, and Acceptance Testing—ensures that the system works as intended and meets user requirements. Together, these processes—detailed design and rigorous testing—are crucial for building a reliable, efficient, and user-friendly enrollment system. By following these methodologies, software developers can ensure the smooth operation of the system, from individual components to full-scale deployment.

**References:**

Terra, J. (2024). *What is Client-Server Architecture? Everything You Should Know.* Simplilearn. Retrieved from [Learning Consultant says... (simplilearn.com)](https://www.simplilearn.com/what-is-client-server-architecture-article)

Visual Paradigm (2023). *Modeling a Client/Server System using UML Packages and Deployment Diagram*. Visual Paradigm. Retrieved from [Modeling a Client/Server System using UML Packages and Deployment Diagram - Visual Paradigm Guides (visual-paradigm.com)](https://guides.visual-paradigm.com/modeling-a-client-server-system-using-uml-packages-and-deployment-diagram/)

Codebun (n.d.). UML diagram for Student Enrollment System. Codebun. Retrieved 10/11/2024 from [UML Diagram for Student Enrollment Management System – Codebun](https://codebun.com/uml-diagram-for-student-enrollment-management-system/)

Geeks for Geeks (2024). State Machine Diagrams | Unified Modeling Language (UML). Geeks for Geeks. Retrieved from [State Machine Diagrams | Unified Modeling Language (UML) - GeeksforGeeks](https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/)

Tsui, F., Karam, O., & Bernal, B. (2018). [*Essentials of software engineering*](https://uagc.instructure.com/courses/137971/modules/items/7033225) (4th ed.). Jones & Bartlett Learning.

Bell, D. (2023). *Explore the UML Sequence Diagram*. IBM Developer. Retrieved from [Explore the UML sequence diagram - IBM Developer](https://developer.ibm.com/articles/the-sequence-diagram/)

Geeks for Geeks (2024). *State Machine Diagrams*. Geeks for Geeks. Retrieved from [State Machine Diagrams | Unified Modeling Language (UML) - GeeksforGeeks](https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/)

Fakhroutdinov, K. (2024). *UML Use Case Diagrams*. uml-diagrams.org. Retrieved from [Use case diagrams are UML diagrams describing units of useful functionality (use cases) performed by a system in collaboration with external users (actors).](https://www.uml-diagrams.org/use-case-diagrams.html)