Week 2 Assignment: UML Design Modeling

Joshua Hohendorf

The University of Arizona Global Campus

CST499: Capstone for Computer Software Technology

Prof. Joseph Rangitsch

August 5th, 2024

Week 2 Assignment: UML Design Modeling

Thorough testing is paramount to ensuring the reliability and functionality of software systems. This process involves multiple levels of testing, each designed to validate specific aspects of the software, ranging from individual components to the entire system. Component testing focuses on verifying the most minor parts of the software, ensuring each unit performs correctly in isolation. Integration testing examines how these components work together, identifying issues that arise from component interactions. System testing evaluates the complete and integrated software to meet specified requirements. Acceptance testing, the final phase, validates the software against user needs and business goals, confirming its readiness for deployment. By understanding each testing level's unique purpose and methodology, developers can build a robust, high-quality software product that meets the needs and expectations of users and stakeholders.

Component Testing focuses on “the functionality and usability of each component are validated separately without integrating it with other components” (Devi, 2023, para. 3). This level of testing verifies that each component functions correctly in isolation. Typically performed by developers, component testing aims to identify and fix bugs within the specific module before integrating it with other components. Unit testing and static analysis are commonly used to validate each element's functionality, performance, and reliability. Ensuring each part works as intended helps prevent issues from arising during later integration and system testing stages. Component testing is the foundation of the testing process, ensuring that the building blocks of the software are solid and reliable.

Integration Testing “exercises two or more software components' ability to function together” (Microsoft Learn, 2023, para. 4). After individual components have been tested, they are integrated into more extensive subsystems or the complete system, and their interactions

are evaluated. This level of testing aims to detect issues related to interfaces and communication between components. Integration testing can be performed incrementally, where components are integrated and tested step-by-step, or in a "big bang" approach, where all components are incorporated simultaneously. The goal is to identify and resolve issues that might not be apparent during component testing, ensuring seamless interaction and data flow between integrated modules. By catching integration issues early, integration testing helps maintain the overall coherence and functionality of the software system.

System Testing validates the complete and integrated software system to meet specified requirements. This level of testing is performed in an environment that closely mimics the production environment. System testing evaluates the software's functionality, performance, security, and overall behavior. This comprehensive testing phase aims to identify discrepancies between the actual and expected outcomes, ensuring the system operates as intended under various scenarios. Testers execute end-to-end test cases covering all functionalities and interactions within the system, verifying that it meets functional and non-functional requirements. System testing ensures that the integrated software works as a whole, providing confidence that the software will perform reliably in a real-world setting.

Acceptance Testing is the final level of testing before the software is released to end-users. This testing phase is to "evaluate the compliance of the system with the business requirements and assess whether it is acceptable for delivery or not" (GeeksforGeeks, 2024, para. 1). Acceptance testing can be divided into alpha and beta testing. The development team conducts Alpha testing internally, while beta testing involves real users in a production-like environment. The primary objective is to validate the software against business requirements and user needs, ensuring it provides the desired functionality and user experience. Successful acceptance testing signifies that the software is ready for production deployment. This level of

testing acts as the final verification step, confirming that the software meets all necessary criteria before going live.

Each testing level plays a critical role in ensuring software quality and reliability. Component testing isolates and fixes issues at the module level, while integration testing addresses interactions between integrated components. System testing comprehensively evaluates the entire system, and acceptance testing ensures the software meets user and business requirements. Together, these testing levels form a rigorous validation process that helps deliver a robust and reliable software product ready to meet the demands of its users and stakeholders. By understanding and applying these testing strategies, developers can ensure the software is well-designed, thoroughly tested, and capable of providing a seamless user experience.

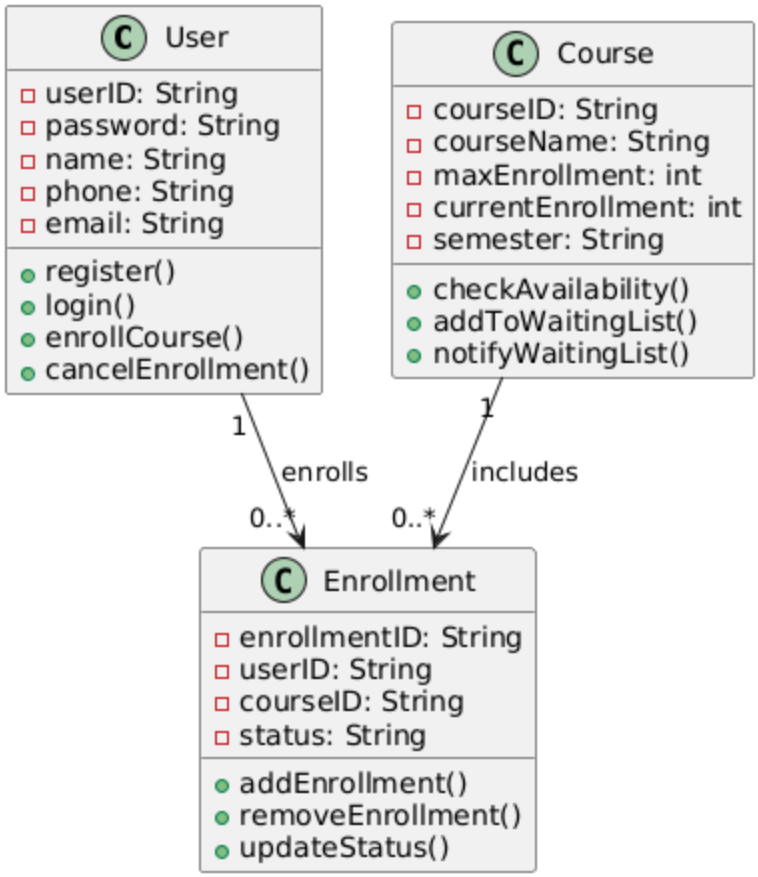
Deciding when to stop testing is critical to the software development lifecycle, balancing thorough validation with practical constraints. “The number of problems found per hour will eventually be so small that the value received from testing starts to diminish” (Tsui et al., 2018, p. 228). Resource constraints, such as time and budget, also play a role in determining when to stop testing, making it essential to prioritize test cases based on risk and impact. The decision to stop testing should be guided by achieving test objectives, meeting quality benchmarks, and ensuring the software is stable and performs as expected in real-world scenarios.

Integrating these levels of testing into the development process creates a robust framework for ensuring software quality and reliability. Component testing establishes a solid base by isolating and validating individual units. Integration testing then verifies that these units work together seamlessly. System testing assesses the entire application's functionality in an environment that mirrors real-world conditions. Finally, acceptance testing ensures the software meets all requirements and is ready for deployment. By following this comprehensive testing

strategy, developers can confidently deliver a high-quality product that meets user needs and stakeholder expectations, ultimately ensuring a successful deployment and a satisfied user base.

Figure 1

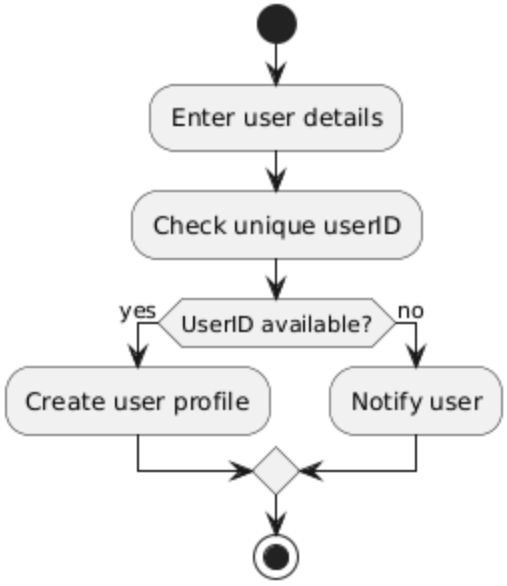
Class diagram of user, course, and enrollments



Note. This UML Model was created using PlantUML.

Figure 2

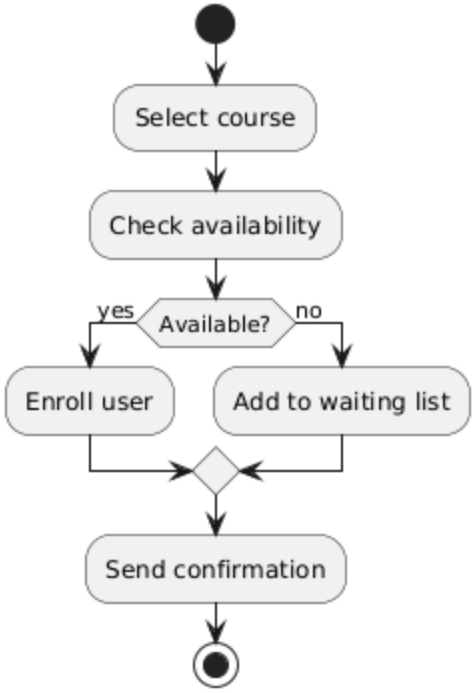
Activity diagram of user registration



Note. This UML Model was created using PlantUML.

Figure 3

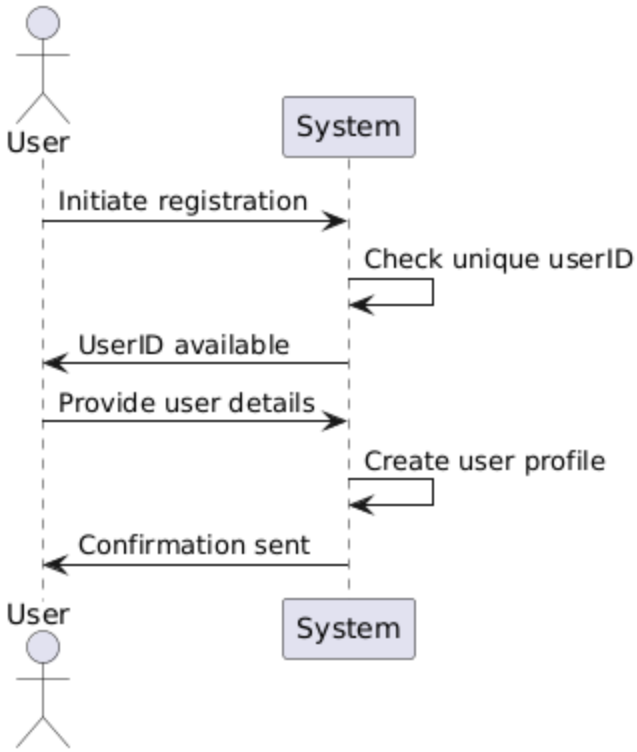
Activity diagram of user enrollment



Note. This UML Model was created using PlantUML.

Figure 4

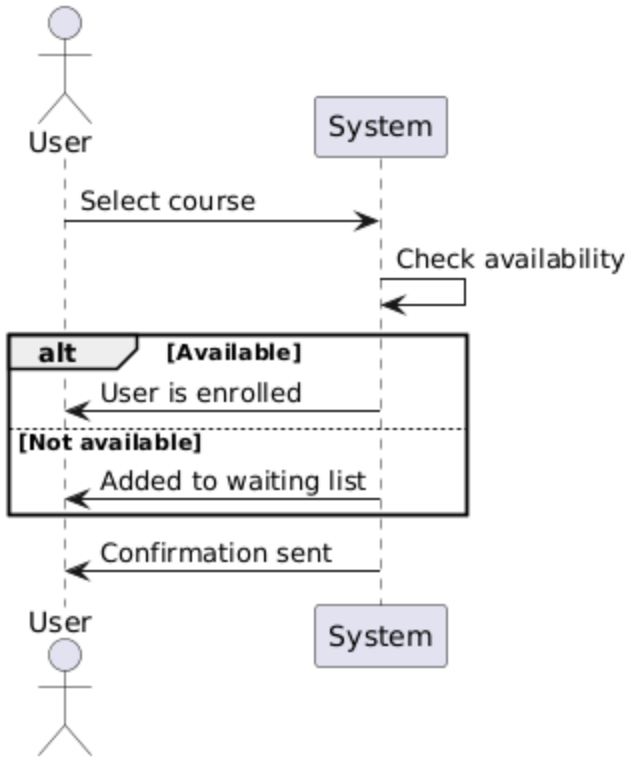
Sequence diagram of user registration



Note. This UML Model was created using PlantUML.

Figure 5

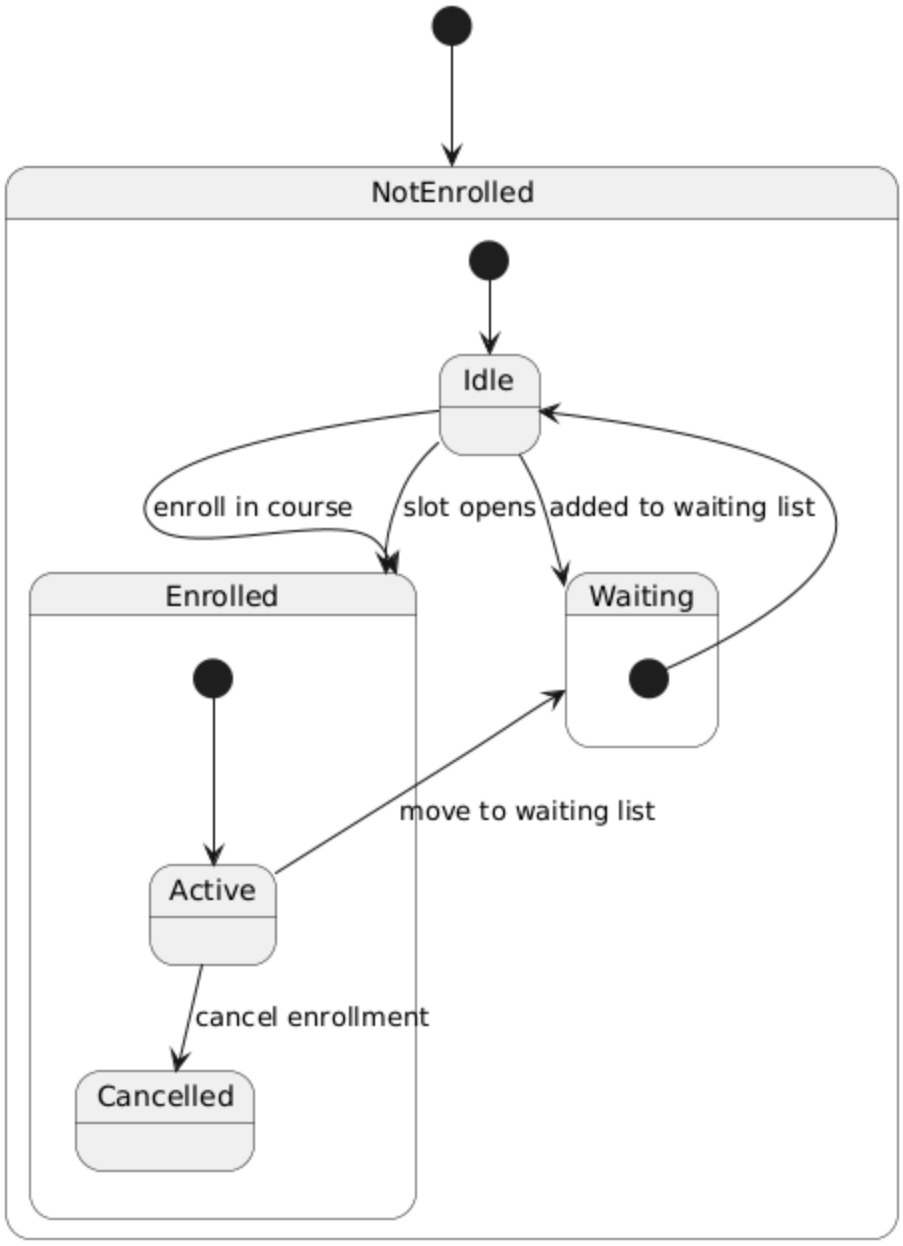
Sequence diagram of user enrollment



Note. This UML Model was created using PlantUML.

Figure 6

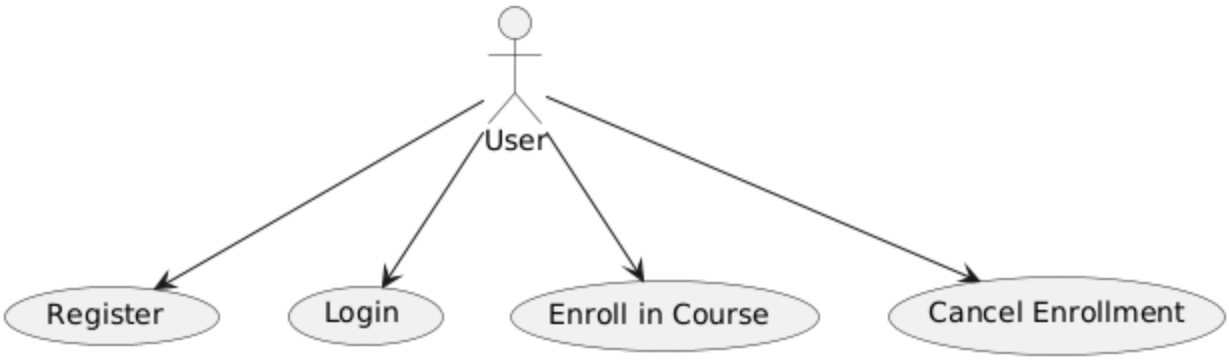
State diagram of user enrollment



Note. This UML Model was created using PlantUML.

Figure 7

Use case diagram of a user of the platform



Note. This UML Model was created using PlantUML.

References

Devi, K., (2023, May 4). *What is Component Testing? (with Examples)*. BrowserStack. https://www.browserstack.com/guide/what-is-component-testing

GeeksforGeeks. (2024, Jun. 12). *Acceptance Testing – Software Testing*. https://www.geeksforgeeks.org/acceptance-testing-software-testing/

Microsoft Learn. (2023, Dec. 16). *Testing in .NET*. https://learn.microsoft.com/en-us/dotnet/core/testing/

Tsui, F., Karam, O., & Bernal, B. (2018). *Essentials of software engineering* (4th ed.). Jones & Bartlett Learning.