**UML Design Model**

Betina Driggers

The University of Arizona Global Campus

CST 499: Capstone for Computer Software Technology

Dr. Charmelia Butler

October 13, 2025

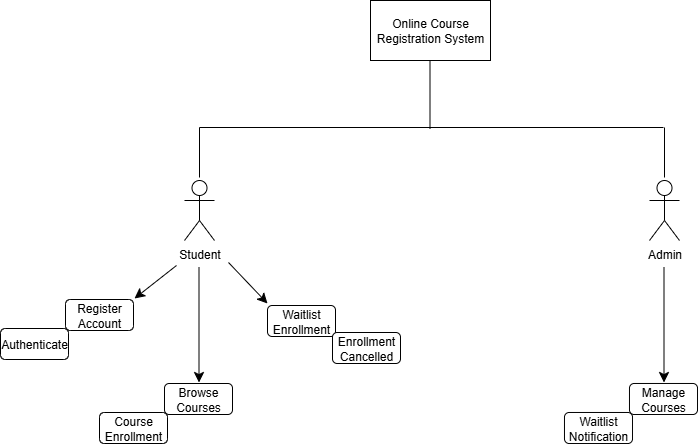
**UML Design Model**

This paper presents Unified Modeling Language (UML) models that translate the Week 1 SRS for the Online Course Registration System into visual representations. The models represent both static and dynamic views, including a use case diagram, class diagram, sequence diagram, activity diagram, and state diagram. The paper outlines a multi-level testing strategy that includes component, integration, system, and acceptance testing to validate the system design and implementation. A coherent UML set, combined with layered testing, helps create a traceable, testable, and maintainable software design (Pressman & Maxim, 2020).

**UML Models**

**Use Case Diagram**

The use case diagram defines the system’s functional interactions by illustrating how students, faculty, and administrators engage with the system. It identifies core functions such as course registration, withdrawal, and waitlist management, which serve as the foundation for defining acceptable test scenarios. The diagram below directly aids acceptance testing by validating that each user interaction aligns with stakeholder requirements (GeeksforGeeks, 2019).

  
Figure 1. Use Case diagram representing student and admin interactions.

**Class Diagram**

The class diagram outlines the system’s structural relationships, including users, courses, and enrollment entities. This representation assists component testing by defining the boundaries of individual classes and their attributes, helping testers isolate units and verify that class behaviors perform as expected under various user/system input conditions (Spillner et al., 2014).

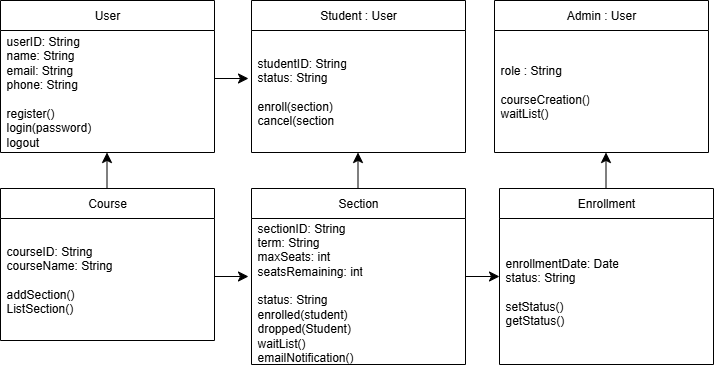


Figure 2. Main classes of the system and relationships between courses, sections, and enrollments.

**Sequence Diagram**

The sequence diagram provides a visualization of the temporal flow of events during a course registration process, such as login, enrollment, and waitlist updates. It is especially valuable during integration testing because it demonstrates message flow and object interactions between modules such as the user interface, database, and notification subsystems (Sommerville, 2016). Testing scenarios derived from this diagram ensure that the system components function seamlessly when combined into one system.

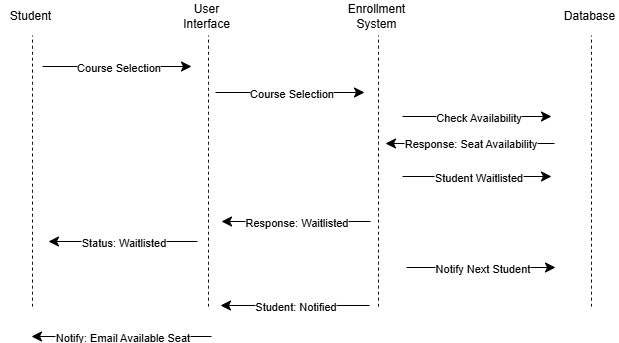


Figure 3. Sequence of course enrollment events, including enrollment cancellation, waitlist addition, and notifications.

**Activity Diagram**

The activity diagram illustrates control flow for processes such as enrolling, dropping, and re-enrolling in courses. It provides a process-level view of parallel and conditional operations, which is useful for validating system testing activities such as performance and usability under different user paths (Pressman & Maxim, 2020).

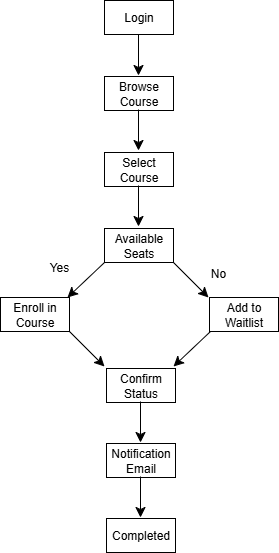


Figure 4. Activity flow for student course enrollment activity.

**State Diagram**

The state diagram represents the lifecycle of a course selection, starting with available classes and ending with either successful enrollment or joining a waitlist due to the course being full. It provides information as to how transitions happen based on user actions or system events. This diagram supports integration and acceptance testing by ensuring the system transitions appropriately and handles state changes without logical or data inconsistencies (Sommerville, 2016).

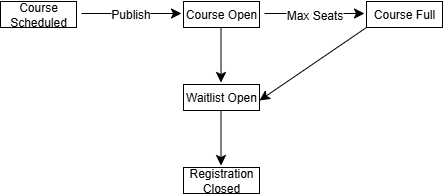


Figure 5. Lifecycle status of course selection.

**Testing Strategy**

Component testing involves unit testing to validate individual classes as services; one example is the Enrollment System. The tests covered within the component testing encompass typical user cases, boundary cases, and any typical error cases that may occur. The class diagram is a great reference for component testing.

Integration testing is a testing exercise that checks the interactions between internal components and external components. For example, it will test the interactions between the user interface, enrollment system, and database. Test scenarios include checking seat availability, adding a student to the waitlist, indicating open seats if a student cancels enrollment, and notification to a student of enrollment status. The sequence diagram highlights how, within integration testing, there are interactions between the various interfaces.

System testing tests the complete system to ensure the system performs as expected under peak load conditions, and the activity diagram illustrates real-world case scenarios. In addition to performance testing, security measures implemented are tested, as well as the usability of the system. Security measures that are tested include access based upon the roles of the user, whether the system has a timeout expiration for inactivity, and whether password requirements are met regarding length and usage.

The main function of acceptance testing is to determine if the system meets the requirements of stakeholders. In addition to stakeholder requirements, acceptance testing also ensures students can locate classes within the system and either enroll or be added to the course waitlist, depending on the status of the course, in a timely manner without errors. The use case and state diagrams directly correlate to acceptance testing as they directly connect to user stories and acceptance criteria.

**Conclusion**

The combined UML set gives a consistent and multi-layered view of the system design while also outlining testing strategies. This ensures the system functions as expected while being integrated into existing systems. This is crucial to help ensure stakeholder expectations are met. By utilizing UML models with an SRS, it is easy to confirm the project stays on track while meeting all expectations, whether that is with the user experience or requirements set by stakeholders or the system itself.

**References**

GeeksforGeeks. (2019, April 29). Acceptance Testing Software Testing. GeeksforGeeks. <https://www.geeksforgeeks.org/software-engineering/acceptance-testing-software-testing/>

Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner’s Approach (9th ed.). McGraw-Hill Education.

Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.

Spillner, A., Linz, T., & Schaefer, H. (2014). [*Software testing foundations: A study guide for the certified tester exam* (4th ed.)](https://uagc.instructure.com/courses/146826/modules/items/7489113). Rocky Nook.