UML Design Modeling

Carlon Bullock

The University of Arizona Global Campus

CST 499: Capstone for Computer Software Technology

Professor: Joseph Rangitsch

April 21, 2025

# Introduction

Educational technology, known for its stout demand for efficient, secure, and scalable registration systems, has never been more urgent. The Online Course Registration and Enrollment System (OCRES) is designed to meet this challenge by offering students, administrators, and faculty a streamlined platform to manage academic enrollments across multiple terms. As course availability, enrollment limits, and user needs fluctuate dynamically throughout an academic year, software systems like OCRES must support diverse functionality and do so with precision and clarity. This level of complexity calls for disciplined software design, which begins not with code but with thoughtful modeling.

The Unified Modeling Language (UML) bridges the gap between system requirements and functional implementation. UML diagrams serve as visual narratives—capturing software entities' static structure, dynamic behavior, and lifecycle transitions in ways that verbal descriptions cannot. According to Tsui, Karam, and Bernal (2022), UML supports communication among developers and traceability and maintainability throughout the software lifecycle. This project uses UML to model critical OCRES processes such as registration, course enrollment, waitlisting, and system-triggered notifications. These diagrams reflect the system's capabilities and anticipate its behaviors under typical and exceptional conditions.

Furthermore, a rigorous testing strategy complements this design, ensuring that each component, interface, and system-wide function behaves as expected. In developing the testing portion of this work, particular attention is given to object-oriented complexity and maintainability. Two related studies by Jangra, Sangwan, and Nandal (2021, 2022) offer compelling insight into how Chidamber & Kemerer (C&K) metrics can be optimized to improve testing effort estimation and quality assurance outcomes. Their 2022 follow-up builds directly on the 2021 review, reinforcing the value of such metrics in planning both unit and integration tests. This continuity stresses the importance of iterative, research-driven design and validation in real-world systems like OCRES. This paper presents a complete UML-based design for OCRES, including use case, class, sequence, activity, and state diagrams. It concludes with a comprehensive testing strategy encompassing component, integration, system, and acceptance testing, all grounded in established engineering principles and supported by scholarly research.

# UML Diagrams

## Use Case Diagram

The use case diagram for the OCRES system identifies the core interactions between external users and the system's functional capabilities. Actors include Students, Applicants, Registrar Staff, and the System itself, each performing distinct roles in initiating or responding to operations such as course registration, profile management, and capacity configuration. This diagram stresses inclusivity and condition-based behavior using UML's <<include>> and <<extend>> relationships. All student-initiated actions must include the login use case to reflect authentication requirements. Similarly, the Send Notification use case is shown as an extension of both Join Waitlist and Cancel Enrollment, capturing the system's conditional behavior in triggering automated messages—a feature highlighted in the system's non-functional requirements for usability and responsiveness (Tsui et al., 2022). Use case modeling provides a high-level functional map of the system, and its simplicity allows stakeholders, including non-technical participants, to verify that all expected interactions are captured early in the design process. This also aligns with agile methodologies described by Gemino, Reich, and Serrador (2020), accentuating shared understanding and communication over technical detail in early planning stages.

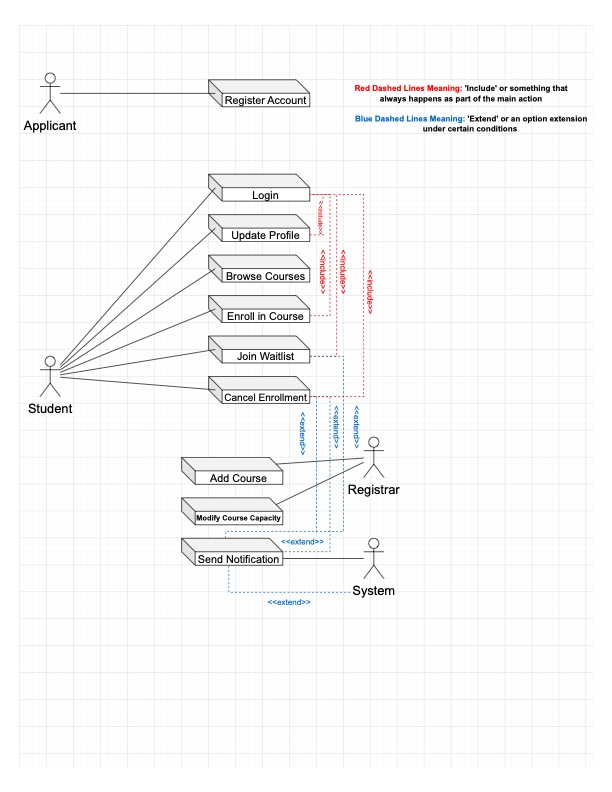


Figure 1. Use Case Diagram for Online Course Registration and Enrollment System (OCRES). Red dashed lines indicate <<include>> relationships; blue dashed lines indicate <<extend>> relationships.

## Class Diagram

The class diagram for OCRES presents the system's static architecture by defining its key object types, attributes, operations, and interrelationships. Central classes include Student, Course, Enrollment, Waitlist, and Registrar, with optional inheritance from a base User class. This structure supports single responsibility and clear entity separation, ensuring that business logic remains encapsulated and testable—a critical factor in maintainability (Tsui et al., 2022). Each class is designed to reflect the real-world roles and data they manage. The Student class holds personal contact information and exposes methods like register() and cancelEnrollment(), while the Course class maintains scheduling, capacity, and roster data through methods like isFull() and addStudent(). The Enrollment class acts as a transactional link between students and courses, and the Waitlist manages overflow using queue-based logic. These object relationships reflect common patterns in academic systems and align well with Chidamber & Kemerer (C&K) metrics, which assess cohesion and coupling to improve class-level testability and change impact (Jangra et al., 2021). What's especially curious here is how even a simple class structure can reveal potential process bottlenecks or opportunities for optimization. Representing Waitlist as a standalone class with its logic opens up test automation and future extensibility, such as implementing priority-based queues or real-time seat notifications—features that are not initial requirements but are made structurally feasible through thoughtful modeling.



Figure 2. A class diagram for OCRES shows primary object classes, attributes, methods, and associations.

## Sequence Diagram

The sequence diagram models a key scenario within OCRES—when a student attempts to enroll in a course. This dynamic view captures the chronological flow of messages exchanged among system objects: Student, Course, Enrollment, Waitlist, and System. It illustrates direct enrollment and the system's contingency behavior when a course reaches capacity. The process begins with the student selecting a course and querying its availability through the isFull() method. Depending on the response, the interaction branches: either proceeding to create an Enrollment record and confirming registration or routing the student to the Waitlist class. In both cases, the System actor issues a confirmation or waitlist notification—demonstrating the system's reactive architecture and event-driven triggers (Tsui et al., 2022).

This diagram gives practical shape to the class relationships previously defined but through a temporal lens. It also reinforces how responsibilities are distributed: for instance, the Course doesn't directly modify the student's state but interacts with enrollment, maintaining cohesion. This type of interaction granularity supports both unit isolation during testing and component substitution—two principles echoed in the C&K metric framework, particularly concerning Response for a Class (RFC) and Coupling Between Objects (CBO) (Jangra et al., 2022). Using conditional branches (e.g., based on course capacity) reflects an essential element of dynamic modeling: the ability to pre-visualize runtime decision points and ensure that alternate flows are supported and testable.



Figure 3. Sequence diagram showing object interactions during student course enrollment in OCRES, including both direct enrollment and waitlist fallback logic.

## Activity Diagram

The activity diagram for OCRES models the workflow of course enrollment from the moment a student logs into the system to the point of successful registration or placement on a waitlist. Unlike sequence diagrams, which emphasize object interactions, activity diagrams focus on process logic—making them ideal for analyzing control flow, user decision points, and system responses. The process begins with a successful login, after which the student selects a course. A decision node then evaluates whether the course has remaining capacity. The system creates an enrollment record and sends a confirmation message if seats are available. If the course is complete, the student is routed to the Waitlist, and a different notification is issued. The diagram ends with a merge at a unified termination point, regardless of the path taken.

Their alignment with agile sprint planning and UI flow visualization makes activity diagrams particularly compelling in design. Tsui et al. (2022) suggest that activity diagrams are handy for communicating functional requirements with stakeholders in early-stage reviews, helping product owners and testers verify logic before implementation. This is especially valuable for systems like OCRES, where minor errors in flow logic—such as mismanaging waitlist conditions—could cascade into significant user-facing problems during registration windows. Additionally, the activity model helps inform test case design by diagramming decisions and their consequences. For example, the "Is course full?" decision node naturally leads to two distinct test scenarios, which should be validated during the integration and system testing phases. This supports Jangra et al.'s (2021) argument that visual workflows help correlate test design with object complexity, streamlining test resource allocation.



Figure 4. An activity diagram representing the workflow of course enrollment in OCRES shows task sequences, decision branches, and system-triggered notifications.

## State Diagram

The state diagram for OCRES models the lifecycle of a student's enrollment status, showing how that state transitions in response to specific actions and system events. Beginning in a Not Enrolled state, a student can move to Enrolled if a course is available or Waitlisted if it is not. Further transitions, such as Dropped or Expired, occur based on user actions or timeouts, offering a full picture of how an enrollment record evolves across its lifecycle. This diagram is especially valuable for clarifying conditional flows and edge cases that may not be immediately obvious in use case or sequence diagrams. For example, the transition from Waitlisted to Expired reflects the system's automated logic to free waitlist spots after a timed offer goes unclaimed—an edge scenario that must be explicitly handled in implementation and testing. Similarly, transitions between Dropped and Reapplied (optional) reflect user-driven re-engagement, which supports flexible academic policies and student autonomy.

State diagrams provide a structured way to visualize compliance and decision logic in systems with complex rules governing eligibility, timing, and automation. Tsui et al. (2022) highlight that state modeling is particularly suited to objects with multiple dynamic phases—such as an enrollment—where knowing "where" the object is in its lifecycle is critical to correct behavior. From a testing standpoint, this clarity enables better coverage of boundary conditions and invalid state transitions, ensuring the system cannot enter an unintended or unstable state (Jangra et al., 2022). Furthermore, the discrete transitions mapped in this diagram can inform test automation frameworks, especially for system and acceptance testing phases. Each state and transition represents a measurable checkpoint or condition to validate, reinforcing traceability between design and validation artifacts.



Figure 5. State diagram modeling the lifecycle of a student's enrollment status in OCRES, including valid transitions such as enrolling, waitlisting, dropping, and expiring.

# Testing Strategy

An effective testing strategy must be tightly aligned with the system's design to validate that it meets its functional and non-functional requirements. For OCRES, testing spans four key levels: component, integration, system, and acceptance. Each stage is designed to systematically reduce risk, catch defects early, and ensure that the final product reflects the expectations documented in the Software Requirements Specification (SRS) and refined through the UML modeling process.

## Component Testing (Unit Testing)

Component testing will focus on individual classes such as Student, Course, Enrollment, and Waitlist, verifying that each behaves according to specification. Unit tests will confirm that methods like enroll(), addStudent(), or cancelEnrollment() operate correctly, return expected values, and enforce boundary conditions—such as enforcing seat limits or detecting duplicate enrollments. The application of C&K metrics, particularly Lack of Cohesion of Methods (LCOM) and Weighted Methods per Class (WMC), will help identify which components are more complex and, therefore, require more thorough unit test coverage (Jangra et al., 2021). The 2022 follow-up study by the same authors expanded on this approach, demonstrating how optimization of these metrics improves effort estimation and reduces defect leakage during component-level testing (Jangra et al., 2022). The development team can prioritize test automation around high-impact areas by analyzing cohesion and coupling early.

## Integration Testing

Integration testing will validate that modules communicate correctly—especially critical in OCRES, where user authentication, course capacity logic, and waitlist behavior are distributed across multiple objects. Key integration points include:

* Student ↔ Enrollment
* Course ↔ Waitlist
* Registrar ↔ Course configuration
* External interface calls to SIS/LMS APIs

The sequence diagram for course enrollment serves as a blueprint for integration test cases, enabling the creation of test scripts that mirror real-world workflows. According to Tsui et al. (2022), proper integration testing should include exception flows, such as attempting to enroll in a full course or triggering a waitlist timeout, to ensure the system behaves reliably under all conditions.

## System Testing

System testing focuses on validating OCRES as a **complete and unified application**, covering all functional requirements defined in the SRS. This includes verifying user flows and security rules, performance expectations, and automated actions like sending notifications. Non-functional requirements from Section 5 of the SRS—such as handling 500 concurrent users, sub-second transaction latency, and in-app notifications—will be tested using simulated load scenarios and stress testing. Metrics like **System Usability Scale (SUS)** and availability (≥99.5%) will be used to quantify performance targets (Gemino et al., 2020). Importantly, system testing will also validate that **invalid state transitions**, such as enrolling after dropping deadlines or accepting an expired waitlist spot, are properly blocked, as illustrated in the state diagram.

## Acceptance Testing

Acceptance testing will involve real users—students and registrar staff—executing scripted scenarios that reflect high-priority use cases, such as registering for a course, updating profile information, or canceling enrollment. Each use case from the UML model will map to an acceptance test scenario, providing traceability from design through validation. This phase emphasizes stakeholder alignment, as outlined in agile frameworks, where user stories and test cases are co-defined to meet evolving expectations (Tsui et al., 2022). Usability and accessibility criteria will be validated per WCAG 2.2 AA standards, while system feedback (e.g., alerts and confirmations) will be assessed for clarity and timeliness.

# Conclusion

The design and development of the Online Course Registration and Enrollment System (OCRES) demonstrate the importance of a structured, model-driven approach to building robust, user-centered software systems. Beginning with a solid requirements engineering foundation, this project translated user needs into precise UML diagrams—each serving a distinct role in clarifying structure, behavior, workflow, and lifecycle. The use case diagram identified functional boundaries and user interactions, while the class diagram established the system's static blueprint. The sequence, activity, and state diagrams brought these abstract elements to life by mapping how objects interact, logic unfolds, and how data evolves. Each of these models informed a testing strategy that is not only thorough but also data-driven, leveraging insights from Chidamber & Kemerer metrics and real-world testing frameworks.

As Jangra et al. (2021, 2022) illustrate in their back-to-back studies, metrics-driven design empowers developers to forecast effort, focus validation resources, and reduce complexity-related errors. This philosophy is echoed throughout the testing methodology used in OCRES, which moves from component testing to full user acceptance, ensuring correctness and usability. Ultimately, the combination of visual modeling and structured testing reflects the best practices in modern software engineering. It brings clarity to complexity, ensures alignment across stakeholders, and increases confidence that the final product will meet the demands of its users. In educational environments where system reliability directly impacts students' academic outcomes, this kind of disciplined design is not just helpful—it is essential.

# Reference:

Gemino, A., Reich, B. H., & Serrador, P. (2020). Agile, traditional, and hybrid approaches to project success: Is hybrid a poor second choice? Project Management Journal, 51(2), 177–194. https://doi.org/10.1177/8756972820973082

Ginikanwa, N. F. (2024). A comparative analysis of agile and traditional project-management methodologies and their impact on information-system development projects: A case study of Nigerian banking system. International Journal of Science and Research, 13(8), 1682–1690.

Jangra, R., Sangwan, O. P., & Nandal, D. (2021). A holistic review on optimization strategies for C & K metrics in object-oriented testing. Proceedings of the 2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT), 362–368. https://doi.org/10.1109/CCICT53244.2021.00073

Jangra, R., Sangwan, O. P., & Nandal, D. (2022). A novel approach for software effort estimation using optimized C&K metrics. Proceedings of the 2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT), 505–510. https://doi.org/10.1109/CCiCT56684.2022.00095

Tsui, F., Karam, O., & Bernal, B. (2022). Essentials of software engineering (5th ed.). Jones & Bartlett Learning.