Software Requirements Specification

for

<UML Design Modeling >

Version 1.0 approved

Prepared by <Nikko Amaya>

<UAGC CST499 >

<Capstone for Computer Software Technology>

<Prof. Charmelia Butler >

<8/12/25>

**Software Testing Levels for a Course Registration System**

**Introduction**

In the development of any software system, rigorous testing is a cornerstone of ensuring that the final product functions as intended, meets user needs, and maintains reliability in real-world conditions. For the course registration system designed in this project, testing assumes particular importance because the system directly manages sensitive student data, regulates access to academic courses, and automates enrollment, waitlisting, and notification processes. A failure in any of these functions could cause significant disruption to students and administrators alike. To mitigate such risks, testing is carried out at multiple levels including component testing, integration testing, system testing, and acceptance testing, each of which addresses quality assurance from a different perspective. This paper explores these levels of testing in detail, emphasizing their role in the lifecycle of the course registration system and how they collectively contribute to the delivery of a dependable application.

**Component Testing**

Component testing, also referred to as unit testing, is the first formal level of testing that occurs once individual modules of the system have been developed. The primary objective of component testing is to validate that each discrete unit of code behaves as expected in isolation. For the course registration system, this means verifying that classes such as Account, Student, Course, Section, Enrollment, and WaitlistEntry operate correctly when separated from other components.

For example, the Account component should successfully enforce unique user IDs, store profile details, and validate credentials against stored information. Similarly, the Section component must reliably track capacity and enrollment counts, ensuring that the current enrollment never exceeds the defined maximum. During component testing, test cases are written to simulate both normal and boundary conditions, such as attempting to register two users with the same ID or incrementing enrollment counts beyond the allowed threshold. By focusing on atomic units of the system, developers can quickly identify logic errors or overlooked conditions within the code, which are easier to correct at this stage than later in the process. Tools such as JUnit for Java or PHPUnit for PHP can automate the execution of unit tests, increasing consistency and efficiency.

In essence, component testing builds confidence that the building blocks of the registration system are stable and trustworthy before they are combined into larger workflows.

**Integration Testing**

After individual components have been validated, the next logical step is integration testing. The purpose of integration testing is to assess whether the interactions between modules function correctly and whether the data exchanged between them is passed and interpreted as expected. Integration testing is particularly critical in a system like course registration, where many interdependent operations must work in sequence.

Consider the enrollment workflow: once a student logs in (Account component), the system retrieves course and section information (Course and Section components), verifies availability (Section component), creates an enrollment record (Enrollment component), and, if necessary, manages the waitlist (WaitlistEntry component) and sends notifications (Notification component). Integration testing would create scenarios that test this sequence holistically, verifying that an enrollment request flows seamlessly across the relevant components.

Integration testing may also use stubs and drivers to simulate components that are not yet fully implemented. For instance, if the Notification service is still under development, a stub might stand in to confirm that the enrollment process still calls the correct method to send an email. Testing integration incrementally, starting with two modules and gradually including others, ensures that errors are easier to isolate. In the context of the course registration system, this level of testing is indispensable because system workflows are inherently crosscutting, and errors often occur not within individual modules but in the way they communicate with one another.

**System Testing**

Once integration testing establishes that components interact properly, the entire system is tested as a unified application. System testing evaluates whether the software meets the requirements defined in the Software Requirements Specification (SRS). Unlike component or integration testing, which focus on technical correctness at the module level, system testing validates the end-to-end functionality, performance, and reliability of the application.

For the course registration system, system testing involves simulating real world user scenarios. Test cases would include a new user registering and creating a profile, students browsing available courses for a given semester, enrolling in courses, being placed on waitlists when courses are full, receiving notifications when seats open, and canceling enrollments. Beyond functional tests, system testing would also include non-functional testing such as performance (e.g., how the system handles hundreds of simultaneous registration attempts at the start of a semester), usability (e.g., whether the user interface is intuitive for students and administrators), and security (e.g., whether passwords are stored securely and whether unauthorized access is prevented).

System testing is executed in an environment that closely mirrors production, allowing evaluators to assess how the entire application behaves under realistic conditions. For example, system tests may reveal whether the registration system properly enforces data consistency when multiple students attempt to register for the same section at the same moment. By uncovering these issues before deployment, system testing ensures the software can be trusted in a live academic setting.

**Acceptance Testing**

The final level of testing is acceptance testing, which is performed to validate that the system meets the expectations and needs of the end users. While previous levels of testing focus primarily on technical correctness, acceptance testing emphasizes business requirements and user satisfaction. In academic settings, acceptance testing would involve both students and administrators, who evaluate whether the system supports their goals effectively.

Acceptance testing typically includes two types: alpha testing and beta testing. Alpha testing is carried out internally by users under controlled conditions, often with developers observing to gather feedback. Beta testing, on the other hand, involves releasing the system to a limited group of actual users in a real environment to capture insights about usability and performance.

For the course registration system, acceptance testing would verify whether students can successfully complete all expected actions: registering for an account, logging in, viewing courses by semester, enrolling in classes, joining waitlists, and canceling enrollments. Administrators would test functions such as creating new course offerings, setting enrollment limits, and managing student rosters. Importantly, acceptance testing evaluates whether the system’s behavior aligns with institutional policies, such as ensuring that no course exceeds its designated capacity or that notifications are timely and accurate.

By the conclusion of acceptance testing, stakeholders determine whether the software is ready for deployment. This stage effectively closes the testing cycle and provides the final assurance that the course registration system can be released with confidence.

**Conclusion**

Testing is not a monolithic activity but rather a structured process that unfolds across several distinct levels, each addressing quality assurance from a different vantage point. Component testing ensures the correctness of individual modules, integration testing validates their interactions, system testing evaluates the application as a whole against specified requirements, and acceptance testing confirms its suitability for real users in real contexts. For a course registration system that directly impacts the academic experiences of students and administrators, rigorous testing across all these levels is indispensable. Each layer of testing builds upon the previous one, creating a comprehensive safety net that minimizes the likelihood of errors escaping into production. Ultimately, this structured approach to testing safeguards not only the technical reliability of the system but also the trust of its users and the reputation of the institution that deploys it.

**USE CASE DIAGRAM**

A diagram of a diagram

AI-generated content may be incorrect.

**CLASS DIAGRAM**

**A diagram of a computer

AI-generated content may be incorrect.**

# Sequence Diagram: Student Lifecycle

This document presents a comprehensive sequence diagram, covering the main system interactions: registration, login, browsing courses, enrollment with capacity and waitlist, and cancellation with notification.

## 1. Registration and Login

1. The Student opens the registration page and creates an account providing necessary information such as userId, password, name, email, and phone.  
2. The Web App UI sends the registration request to the AuthService.  
3. AuthService queries AccountStore to check for duplicate userIds.  
4. If the userId is available, AccountStore creates a new account and returns confirmation.  
5. AuthService requests NotificationService to send a welcome email, which is dispatched through the EmailGateway.  
6. The UI informs the Student of successful registration.  
7. If the userId is a duplicate, the system returns an error and prompts the Student to use a different ID.

8. For login, the Student submits credentials via the UI.  
9. UI forwards credentials to AuthService for authentication.  
10. If valid, AuthService returns the Account and the UI shows a successful login message.  
11. If invalid, an error message is displayed.

## 2. Browsing Courses

1. The Student selects a term (spring, summer, fall) in the UI.  
2. The UI queries the CourseCatalog for courses available in that term.  
3. CourseCatalog returns a list of courses.  
4. The Student requests to view sections for a chosen course.  
5. The UI requests active sections from the CourseCatalog.  
6. CourseCatalog returns sections with capacity and enrollment details.

## 3. Enrollment

1. The Student chooses a section and requests enrollment through the UI.  
2. UI forwards the request to the RegistrationService.  
3. RegistrationService queries the SectionStore for details.  
4. If seats are available, RegistrationService:  
 - Creates an enrollment record in EnrollmentStore.  
 - Increments the section enrollment count in SectionStore.  
 - Notifies the NotificationService to send a confirmation email.  
5. The UI shows enrollment success to the Student.  
6. If the section is full, RegistrationService:  
 - Adds the Student to the WaitlistStore.  
 - Requests NotificationService to send a waitlist confirmation email.  
7. The UI shows the Student their waitlist status.

## 4. Cancellation and Waitlist Promotion

1. The Student cancels an enrollment via the UI.  
2. The UI sends the request to RegistrationService.  
3. RegistrationService updates the enrollment status in EnrollmentStore to 'Canceled'.  
4. SectionStore decrements the enrollment count for the section.  
5. RegistrationService queries WaitlistStore for the first waitlisted student.  
6. If a student exists on the waitlist:  
 - RegistrationService creates a new active enrollment in EnrollmentStore.  
 - NotificationService sends an email notifying the waitlisted student that they have been enrolled.  
7. If no waitlist entries exist, the process ends.  
8. The UI confirms cancellation to the Student.

**REFERENCES**

Tsui, F., Karam, O., & Bernal, B. (2018). [*Essentials of software engineering*](https://uagc.instructure.com/courses/149470/modules/items/7628542) (4th ed.). Jones & Bartlett Learning.

GeeksforGeeks. (2025). *Component software testing*. <https://www.geeksforgeeks.org/software-engineering/component-software-testing/>

Powell, P., & Smalley, I. (2025). *What is integration testing?* IBM. from <https://www.ibm.com/think/topics/integration-testing>

BrowserStack. (2025, May 29). *What is system testing? (Examples, use cases, types)*. <https://www.browserstack.com/guide/what-is-system-testing>

Testsigma. (n.d.). *Acceptance Testing: What, Why, Types & How to Do*. <https://testsigma.com/guides/acceptance-testing/>