**UML Design Modeling**

Austin Blair

The University of Arizona Global Campus

CST499: Capstone for Computer Software Technology

Joseph Rangitsch

2-8-2025

**Introduction**

When designing software, there are a multitude of steps in the software development lifecycle (SDLC). Two key concepts that are essential to delivering effective software are UML diagrams and software testing. UML diagrams offer explicit, representative models that capture the system's primary design requirements. Software testing evaluates software quality, ensures functionality, and delivers products that satisfy stakeholder requirements. Within software testing, there are four testing levels that coincide with the development phases of the SDLC, which is often referred to as the V-model. The V-model displays that development and testing tasks are corresponding activities of equal importance (Spillner, 2014). The four levels of software testing are component, integration, system, and acceptance. Each of these will be discussed further in the following sections.

**UML Diagrams**

Different modeling methods can be utilized in software design, but the standard modeling language is UML. UML, Unified Modeling Language, is a general-purpose modeling language that defines a standard way to visualize a software system's design, behavior, and structure (GeeksforGeeks, 2025). UML diagrams are broken into two categories: structural and behavioral. For the design of the Online Student Enrollment System, the structural subtype Class-Diagram was utilized; see Figure 1 below. Class diagrams depict the static structure of a system by A screenshot of a computer generated diagram

Description automatically generatedshowing the system's classes, methods, and attributes (GeeksforGeeks, 2025). Of the behavioral category of UML diagrams, the State Machine, Activity, Use-Case, and Sequence diagrams were used to represent the dynamic aspects of a system. The dynamic aspects illustrate how objects interact and behave over time in response to events (GeeksforGeeks, 2025). The State Machine diagram, as seen in Figure 2, represents the condition of the Online Student Enrollment system at fixed instances in time. A diagram of a course

Description automatically generated

**Figure 1: Class Diagram**

**Figure 2: State Machine Diagram**

A screenshot of a computer screen

Description automatically generatedThe Activity diagrams represent the flow of control in the Online Student Enrollment system. Figure 3 depicts the Login and Registration flow, and Figure 4 illustrates the Course Enrollment flow within the system.

A diagram of course

Description automatically generated

**Figure 4: Course Enrollment Activity Diagram**

**Figure 3: Login/Registration Activity Diagram**

The Use-Case diagram, as seen in Figure 5 below, illustrates the system's functionality by showing the different actions permitted by each user of the system. The last UML model for the online enrollment system is represented in Figure 6. The Sequence diagram displays how and in what order, the objects in the Online Enrollment System function.

A diagram of a course

Description automatically generated

A diagram of a course

Description automatically generated

**Figure 5: Use-Case Diagram**

**Figure 6: Sequence Diagram**

**Component Testing**

Component testing, also known as unit testing, is used to verify whether the mode of operation of the component, module, program, object, or class is functional or non-functional (Homès, 2012). These types of tests are usually executed on components that can be tested individually and with access to the source code. Component testing is, in most cases, performed by the developers themselves and is often referred to as a developer test (Spillner, 2014).

**Integration Testing**

Integration testing verifies if groups of components interface in the way specified by the technical system design. This second level of testing includes interfacing with the operating system, file systems, databases, hardware, and software (Homès, 2012). The main objective of integration testing is to reveal interface problems and conflicts between integrated parts.

**System Testing**

System testing is used to detect software failures and ensure that it corresponds to the requirements and specifications provided by the customer (Homès, 2012). A successful system test would validate how well the complete system meets the specified functional or non-functional system requirements. It is important to note that the system test does not test only the system but also the user documentation, such as system and user manuals and training material (Spillner, 2014).

**Acceptance Testing**

The last level of testing is the Acceptance test. Acceptance testing is used to obtain customer and user approval of the software system. These tests are mainly black-box tests executed by users or customer representatives and augmented by testers (Homès, 2012). There are four typical types of acceptance testing: contract, user, operational, and field testing. Contract acceptance testing is used if customer-specific software was developed. User acceptance testing is recommended if the customer and user are different. Operational acceptance testing ensures the system's acceptance by system admins (Spillner, 2014). Lastly, field testing identifies influences from users' environments that are not entirely known or specified and eliminates them if necessary. This last level of testing primarily corresponds to the requirement analysis phase of the SDLC. Still, it can also occur during software design, integration with other software, or the acceptance of new functionalities or features (Homès, 2012).

**Conclusion**

The SDLC consists of phases: planning, design, testing, and maintenance. During the design phase, UML diagrams are utilized to effectively represent the primary design requirements of the system. Once all requirements are represented and agreed upon, and the design is completed, testing ensures that the software system meets quality and functionality standards while delivering a product that satisfies stakeholder requirements. These two phases within the SDLC are essential to the overall effectiveness and quality of the final deliverable.

**References**

GeeksforGeeks. (2025, January 2). Unified Modeling Language (UML) Diagrams. <https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/>

Homès, B. (2012). Fundamentals of Software Testing. ProQuest Ebook Central. <https://ebookcentral.proquest.com/lib/ashford-ebooks/detail.action?docID=1120766>.

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/137170/modules/items/6985762). Rocky Nook.