**UML Design Modeling**

Mitchell Rozonkiewiecz

University of Arizona Global Campus

CST499: Capstone for Computer Software Technology

Dr. Butler

October 14, 2024

Testing is a critical component of software development when designing a new system. Several areas of testing should be part of the implementation plan. The most common testing types include component, integration, system, and acceptance testing. Let us review the goal of each phase, how it benefits the application, and which UML diagram can assist with this phase.

When initially coding a new module, you must perform unit testing before it can be used in the application. “Generally, we speak of software units or components. Testing of a single software component is therefore called component testing” (Spillner et al., 2014, pg. 42). Component testing aims to verify that the function or method performs as desired at the lowest level before integrating into the application for higher-level functionality. Today’s organizations often use Test Driven Development (TDD) or Behavior Driven Development (BDD) to review the module and determine how to test and certify. This is important as it ensures that not only is the happy path good, but that bad data does not cause abends. This testing can be approached from either a black-box or a white-box point of view. When testing using a black-box approach, you only know the parameters passed into the function. This allows testing using values in and out of range and good and bad data types. If you can then test using the white-box approach, you get to see how the internals work and design tests that could make it fail by knowing this information. A UML Class diagram can assist with this testing as it allows for the design and implementation of black-box testing. See Figure 1 for a class diagram for the Class Registration Application.

**Figure 1**

UML Class Diagram

A diagram of a server

Description automatically generated

The next step in testing is when a few units, or components, are assembled and tested together. The type of testing that verifies modules working together is called integration testing. Before starting integration testing, the individual units must pass their component tests. This ensures that if issues are found, they are due to module interaction and not fundamental module issues. This step aims to find faults in the interfaces when multiple components interact with each other. Even when component tests are successful, interface issues can be found. The value of this step should not be minimized or skipped. “Furthermore, in a diverse, large-scale organization setting, the testing approach has to be open for an integration into different automation driven delivery chains” (Poth et al., 2022, pg. 202). UML State and Sequence diagrams can assist with this testing. See Figure 2 and 3 for a state and sequence diagram for the Class Registration Application.

**Figure 2**

UML State Diagram

A diagram of a flowchart

Description automatically generated

**Figure 3**

UML Sequence Diagram

A diagram of a user flow

Description automatically generated

The following testing step is assembling all components into the final application. This assembly of all modules is tested to verify that the system meets the initial design requirements gathered from the user. This set of tests is called a systems test. Why is this needed if all the component and integration tests are successful? The testing discussion so far has concentrated on meeting technical specifications. System testing is different in that it tests the application from the users’ point of view, including non-functional requirements. For example, an area often overlooked is the interface to the operating system environment. Development environments can and do run leading-edge copies of the operating system and fixes. When deployed into production corporate environments, these usually are more reserved as they prefer stability over the latest feature. UML Use Case diagrams can assist with this testing. See Figure 4 for a use case diagram for the Class Registration Application.

**Figure 4**

UML Use Case Diagram

A diagram of a product registration app

Description automatically generated

The last testing step is called acceptance testing. All the prior tests for the application are under the control of the software designer. The application customer performs acceptance testing. Test scenarios are documented and executed after installing the application and reviewing the features needed. This will determine if the designer implemented the desired functionality and if the complete application was packed and shipped. This installation will assist in detecting if any modules are missing from the install package.

In conclusion, we reviewed each area of testing and the associated goal to be accomplished. We then review possible approaches, such as black-box and white-box, to see the benefits of the different approaches. Finally, we included different UML diagrams, including a Class, Sequence, State, and Use Case, to see how they can assist with that testing phase.

**References**

Poth, A., Rrjolli, O., & Riel, A. (2022). Integration- and System-Testing Aligned with Cloud-Native Approaches for DevOps. *2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C), Software Quality, Reliability, and Security Companion (QRS-C), 2022 IEEE 22nd International Conference on, QRS-C*, 201–208. <https://doi.org/10.1109/QRS-C57518.2022.00038>

Spillner, A., Linz, T., & Schaefer, H. (2014). *Software testing foundations: A study guide for the certified tester exam* (4th ed.). Rocky Nook. <https://uagc.instructure.com/courses/134276/modules/items/6830274>

Tsui, F., Karam, O., & Bernal, B. (2018). *Essentials of software engineering*. Jones & Bartlett Learning. <https://uagc.instructure.com/courses/137971/modules/items/7033225>