**UML Design Modeling**

Dennis Walters

University of Arizona Global Campus

CST 499: Capstone for Computer Software Technology

Dr. Amjad Alkilani

November 15, 2021

**UML Design Modeling**

The deployment of good software consists of many steps beginning with requirements gathering, then design modeling. According to Booch (2005), “Modeling is a central part of all the activities” involved in getting good software developed and deployed. Modeling provides a visual of the system’s architecture and allows the designer to have complete control of that architecture. It also provides a better means by which to understand the system being built and can uncover possibilities for reuse and simplification of code. Finally, models aid in risk management.

**Component Testing**

The component testing level tests individual software components to verify that they each meet the corresponding requirement fully and completely. This is the first attempt at testing the software components. The components, depending on the programming language, could be referred to as modules, units, or classes (Spillner et al, 2014). The objectives of component testing are to prove, or disprove, functionality, robustness, efficiency, and maintainability of the component. To do this, a test driver program, like Postman, is used. There are several test strategies to choose from when conducting component testing. White box testing requires the source code to be analyzed in addition to the testing. In black box testing, input is provided to the component and the output is collected. Test-driven, or “Test First”, development involves designing and automating tests prior to beginning programming of the components. This is an iterative approach as the code is tested with the previously developed test cases, then the code is improved until it passes the tests.

**Integration Testing**

The integration testing level, which occurs after component testing has been completed, is used to check whether sets of components integrate and interact with each other as specified in the design of the technical system. Before components can be integrated, they have to have passed their individual component testing first. Groups of components are then integrated to form larger structural units and subsystems (Spillner et al, 2014).

To integrate the components there are four generic strategies the test manager can choose from to follow; Top-down integration, Bottom-up Integration, Ad hoc integration, and Backbone integration. Top-down and Bottom-up integration work as they sound, although they should only be used in situations where the system design is structured hierarchically. Ad hoc integration involves integrating components as development and component testing are completed. Backbone integration involves building a backbone to add components to gradually. Non-incremental integration, also called big bang integration, should be avoided as much as possible. This integration strategy involves waiting until all components have been developed and tested, then throwing them all together in one step then testing.

The structural units and subsystems are then tested to ensure that all the components that they are comprised of collaborate with each other without any issues. Integration testing exposes any faults in the interfaces as well as in the interaction between the integrated components. This level of testing is necessary to ensure that the interaction between components works as designed.

**System Testing**

The system testing level tests all components as a whole to ensure that the specified functional and non-functional requirements have been met (Spillner et al, 2014). This step only occurs after integration testing has been completed and all the subsystems have been integrated together into the complete system. System testing uses all the documents and information that describe the system such as specifications, system requirements, user manuals, and risk analyses if available.

The intent of system testing is to test all of the integrated components in an environment that represents as closely as possible the production environment that will be used by end users. Instead of using test drivers, like Postman, the software is installed on a complete system including hardware, system software, any necessary device drivers, networks, and any external system connections needed such as a database. In addition to testing the system itself, items like system and user documentation, configuration settings, and system configuration optimization are also covered and tested. It is imperative that system testing not be conducted on an end user’s system as if there are any issues the user’s environment could be compromised.

From this testing documentation of failures from inconsistent, incorrect, or incomplete implementation of the requirements should be created. This should also include any forgotten or undocumented requirements that have been identified during the process. There are some issues that can arise in the practice of system testing. If no requirements are provided, then there is no way to truly know if the system is behaving in the manner that it should. Decisions can be overlooked and not appear again until testing occurs causing additional time and cost to development and testing. And, ultimately, the project could completely collapse and fail due to the afore mentioned issues.

**Acceptance Testing**

The acceptance testing level involves the end customer and verifying that the system meets the specified requirements and end user needs and expectations. All of the previous levels of testing have involved testing the functional and non-functional requirements from the development and testing perspective (Spillner et al, 2014). Acceptance testing involves focusing on the customer and user experience perspective and may be the only testing that involves the customer and/or end user. Earlier testing levels could also involve acceptance testing such as a component’s usability during component testing or checking new functionality before system testing.

There are four typical forms of acceptance testing that may be used during this level of testing. Contract acceptance testing takes the requirements outlined in the contract and verifies that the software meets all of them. User acceptance testing uses business processes and typical usage scenarios, also called use cases, to verify that the software meets user requirements and expectations. Operational testing involves testing backup and restoration processes, disaster recovery, user management, and any possible vulnerabilities in security to assure acceptance by system administrators. Field testing, also known as Alpha and Beta testing, involves providing the software to a select group of end users to test usability in the end user’s environment as they are not necessarily ones that have been previously tested for. Finally, there is a newer term used to describe internal testing by the makers of the software, “Dogfood” testing. Basically, the idea is that if you made it, you should try using it yourself first.

**Figure 1**

***User Registration and Login Activity Diagram***

Diagram

Description automatically generated

**Figure 2**

***Course Enrollment Activity Diagram***

Diagram

Description automatically generated

**Figure 3**

***Class Diagram***

Graphical user interface, website

Description automatically generated

**Figure 4**

***Entity Relationship Diagram***

**Graphical user interface, application, Teams

Description automatically generated**

**Figure 5**

***Sequence Diagram***

Text

Description automatically generated

**Figure 6**

***State Diagram***

Diagram

Description automatically generated

**Figure 7**

***Use Case Diagram***

Diagram

Description automatically generated

**References**

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The unified modeling language user guide* (2nd

ed). Addison-Wesley.

Spillner, A., & Linz, T. (2021). *Software Testing Foundations: A Study Guide for*

*the Certified Tester Exam* (4th ed.) [E-book]. Rocky Nook.

Tsui, F., Karam, O., & Bernal, B. (2018). *Essentials of software engineering* (4th ed.). Jones &

Bartlett Learning. Retrieved from <https://platform.virdocs.com/r/s/0/doc/581911/sp/175231953/mi/563314074?cfi=%2F4%2F4>.