UML Design Modeling

Brandyn Tweedly

The University of Arizona Global Campus

CST 499: Capstone for Computer Software Technology

Professor Amr Elchouemi

June 27, 2022

UML Design Modeling

UML designs, or Unified Modeling Language designs, are meant to visualize multiple different aspects of a software project and can be completed in multiple different ways. A UML design has been included at the end of this document for analysis as it pertains to the software requirements that were previously gathered pertaining to a student information system and course scheduling system. A system such as this requires testing to ensure reliability for an end user. Testing the software for functionality based off of the requirements that were previously engineered ensures compliance and quality with the product and alignment with the business objectives and stakeholders or clients that are involved with the project. The different levels of testing that will be explored in this document are component testing, integration testing, system testing, and acceptance testing.

To begin at the “lowest” level of testing, this document will examine component testing, also known as unit testing. This type of test covers the foundational units of a program or software on an individual level, such as in Object Oriented Programming, or OOP, testing a class or method for functionality and meeting the associated requirements. Per Sillner (2014) "The main characteristic of com-ponent testing is that the software components are tested individually and isolated from all other software components of the system. The isolation is necessary to prevent external influences on components." Testing for functionality in this case refers to the completeness of the unit or component regarding the associated requirement and any inputs or outputs that are a part of that component’s functionality.

Integration testing moves the testing process up a level from component testing. After all component tests have been completed and any bugs or defects in the code have been addressed and fixed or remedied, the components can then begin testing as groups, this process is what is

referred to as integration testing. By combining the components into groups for testing, the integration process will now refer to those groups as subsystems. Pertaining to the development and scope of integration testing and moving individual components into subsystems after unit testing has been completed, Bures (2020) states “in these systems, a more extensive set of various protocols on different networks and application levels can be integrated together, and seamless integration has to be maintained…increased necessity to employ techniques testing correct functionality of integration interfaces and interoperability with different configurations.” While unit testing may have been successful, that does not always indicate that integration testing will be successful, as stated by Spillner (2014) "Even if a complete component test had been executed earlier, such inter-face problems can still occur. Because of this, integration testing is necessary as a further test level. Its task is to find collaboration and interoperability problems and isolate their causes."

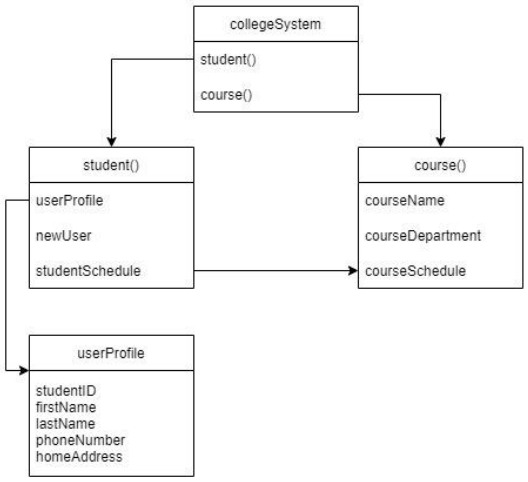
With component testing and integration testing now completed and subsystems established, we can now move into the implementation and integration of those subsystems into the larger system itself. Tsui (2018) states that, in regards to systems, "The use of assertions to promote defensive programming is explored along with the implementation activities of planning, building, projecting, tracking, reviewing and updating the code. These activities enable the realization of the system." The system test is not only the next step in testing, but a necessary one, even though the previous two tests have been run and assumably caught and addressed any bugs or defects in the code base. Per Spillner (2014) "System testing checks if the integrated product meets the specified requirements…The system test, though, looks at the system from the perspective of the customer and the future user." This type of testing also happens in a test

environment that is more closely modeled to the environment in which the system would actually be used, such as testing on certain hardware or operating systems.

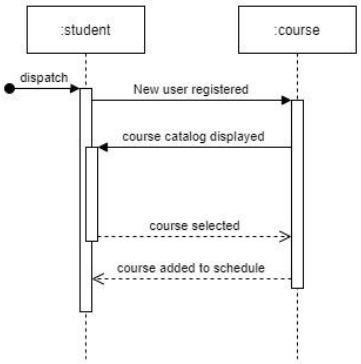
The fourth and final level of testing is the acceptance test. After the system test has been completed and verified that it behaves as it should when being utilized by an end user, a software will be transitioned to the acceptance, or user testing phase. Spillner states that “"the focus is on the customer’s and user’s perspective. The acceptance test may be the only test that the customers are actually involved in or that they can understand." The purpose of this test is typically to get the product into the hands of the client or a test group that would be representing the client and have that test user use the system or product in the way it was intended to be used to ensure that all requirements have been met and the product is operating as stated with no defects or bugs in the system.

With the four levels of testing being explored in this document as they related to requirements, and how those requirements are outlined and visualized in a UML diagram, a test plan can be established for a software product or project. The examinations of testing in this case are at a foundational level and the levels themselves include finer details and processes relating to how testing takes place and the systems or tools that are utilized. The four test levels, component, integration, system, and acceptance testing are all integral processes and must be a priority in any project.

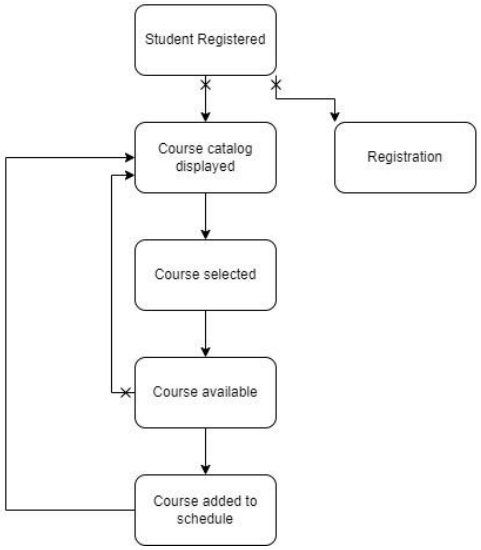
Class Diagram



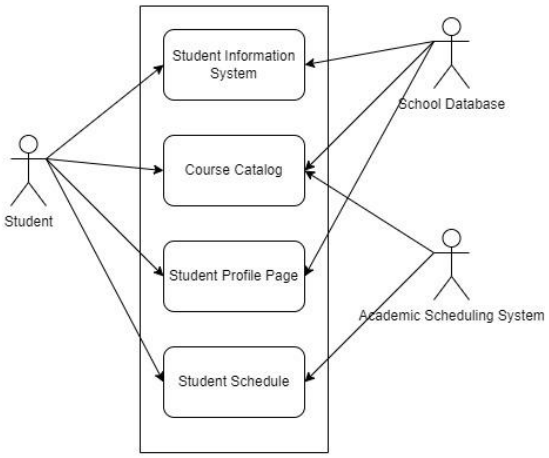
Sequence Diagram



State Diagram



Use Case Diagram



References

Bures, M., Klima, M., Rechtberger, V., Bellekens, X., Tachtatzis, C., Atkinson, R., & Ahmed, B. S. (2020). Interoperability and Integration Testing Methods for IoT Systems: a Systematic Mapping Study.

Spillner, A., Linz, T., & Schaefer, H. (2014). Software testing foundations: A study guide for the certified tester exam (4th ed.). Rocky Nook.

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