**Week 2 Assignment - UML Design Modeling**

Laura Harrington

University of Arizona Global Campus

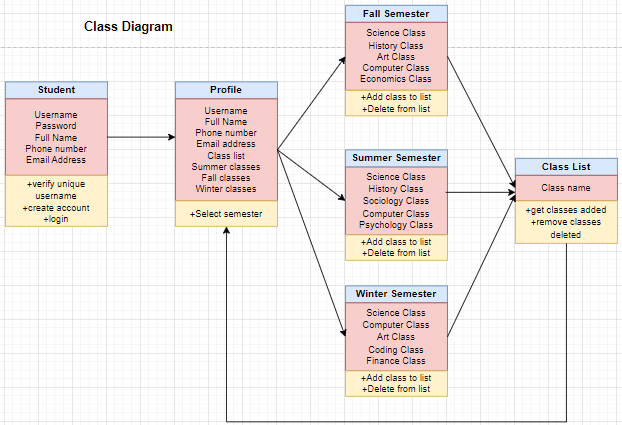
CST499: Capstone for Computer Software Technology

Instructor Rangitsch

May 27th, 2024

A software development team’s main goal is to create quality software that satisfies all agreed upon requirements. This takes a certain level of planning no matter what development methodology is being used. One way to ensure requirements are met throughout the system is creating a series of UML model diagrams. They offer a guide for how a system should be structured as well as specific behaviors expected from the system. These diagrams include, but are not limited to, class diagrams, sequence diagrams, activity diagrams, state diagrams, and use case diagrams (Tsui et al., 2018). These diagrams can also be used to plan a series of test cases to be performed at every stage of development. Testing is an important part of software development as it mitigates risks of delivering code that could be incomplete or have errors. When thorough testing is performed at multiple stages it ensures that the software meets specifications and is ready to be used by the end users (Tsui et al., 2018).  
**Figure 1**

**Class Diagram**



Class diagrams give a visual representation of the main objects in a system along with the relationship between one another and what those objects do in the system (*Rational software architect 9.7.0*, 2023). Class diagrams can also give a good idea of the components of a system, which sets up for component testing. Component testing is utilized to test the usability and behavior of each component individually (*Component software testing*, 2019). A great way to do this is to test by class, as each separate object of a system is being coded into existence and worked on a series of tests can be performed to ensure each one is working before it is implemented as a whole system (*Component software testing*, 2019). This testing can be performed by the developer or a partner developer. If errors are found, they can be fixed before they are combined and the functionality of their relationship to one another is tested.   
**Figure 2  
Sequence Diagram**

A diagram of a sequence diagram

Description automatically generated

**Figure 3  
State Diagram**

A diagram with text and words

Description automatically generated

The user experience is heavily considered throughout the course of building diagrams and performing a series of tests. Specifically, what is the user needing to accomplish while using the system being designed? What pieces of the system should each action need to pass through? Those questions are answered and put into a visual format with the sequence diagram. It records each action a user could take and what the system might do in response to that action. It shows how the user will initiate an action, what part of the system it will reach and the response they should get (Tsui et al., 2018). In Figure 2 it shows the sequence of events when creating a user account, logging in, and choosing classes to attend. In figure 3 it shows a state diagram which goes hand in hand with the sequence diagram, as it takes the actions initiated by the user it shows the states the system will go into as the user interacts with the system (Tsui et al, 2018).

After all pieces of the system are combined the integrated functionality of a system can be tested. This is also a series of testing that is performed on the system in its entirety, this level of testing is called integration and system testing (Tsui et al., 2018). It is important to ensure that every separate component is behaving as requirements specified before they are combined and tested again. The sequence diagrams can be utilized to pull specific test case scenarios. An example of a test case is; when a new user attempts to submit a username already saved in the database does the error message display as expected? This makes sure that the action taken by the user follows the sequence of events and passes through the correct parts of the system, in this specific test it would move through all levels. State diagrams can also assist with complete well-rounded integration and system testing as the test cases created from that diagram will cover not only functionality of the system but ensuring that the various states the system should be in are also correct (Tsui et al, 2018). This testing covers a lot of the use case scenarios but should be performed prior to any official acceptance testing performed by stakeholders or users.

**Figure 4  
Activity Diagram**

A diagram of a computer program

Description automatically generated

**Figure 5  
Use Case Diagram**

A diagram of a person with text and green hexagons

Description automatically generated

The final type of testing that should be performed is acceptance testing. By now the system and all its components have been tested at code level and run through lower-level environments. In a lower environment as close to the real production environment the key stakeholders and users should perform acceptance testing (Tsui et al., 2018). This testing is formal and test cases should be pulled from use case and action diagrams. This is because these tests have nothing to do with the code, it is all about the user and how they will interact with the system. After the tests are performed the stakeholders, customers, and users can accept the software and it can either be implemented or purchased (Tsui et al., 2018).

UML model diagrams are an important part of building top quality software for customers. They not only provide a visual structure of the system and all its components, but also a way to see the various sequences of events that can take place, and states the system is expected to be in while a user interacts with the system (*Rational software architect 9.7.0*, 2023). These diagrams also aid in testing software on all levels while it is being developed. Testing can identify any failures or problems early in the project and they can be fixed before they are implemented. The use of UML model diagrams to build test cases can mitigate risks to the system and user experience once it is released.

**References**

*Component software testing.* (2019, August 14). GeeksforGeeks. <https://www.geeksforgeeks.org/component-software-testing/>

*Rational software architect 9.7.0.* (2023, September 21). Ibm.com. <https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-uml-models>

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