UML Design Modeling

Jamain Hughes

The University of Arizona Global Campus

CST 499: Capstone for Computer Software Technology

Professor Dr. Amjad Alkilani

January 8, 2024

Software engineering involves careful planning in order to ensure that customers get exactly what they want within a specific timeframe. In my exploration of the Student Registration System, as outlined in the Software Requirements Specification (SRS) document, I defined the system's scope and objectives. The primary goal is to streamline the course enrollment process for students. The student registration system should have a user-friendly interface that enables account creation, course registration, waiting list management, and enrollment cancellations. To gain a deeper understanding of the functional and non-functional requirements, I used unified modeling language (UML) diagrams, including use case diagrams, class diagrams, sequence diagrams, state diagrams, activity diagrams, and a database schema. These UML representations serve as foundational blueprints, guiding the analysis, design, and eventual implementation stages of the software system's life cycle.

As outlined in my SRS document from my week 1 assignment, I defined the scope of the Student Registration System. The scope of this system is defined as:

This project is essentially a student registration a student registration system. The overall goal of this is to allow students to easily manage their course schedules. Students will be able to register as a new student and create their account and profiles. Students will also be able to enroll themselves onto courses if they are available during the respected semester. If the desired course is not available, students can add themselves onto a waiting list. Students will also be able to cancel their enrollment from any course that they are enrolled in, and the system should inform the first in the waiting list that they can enroll into the class. This project aligns with the business objectives of holding onto students, improving the student’s online college experience, and reducing costs while increasing the speed of the college registration process compared to manually registering students.

In order to gain a better understanding of the entire system I have illustrated my system from the perspective of the Unified Modeling Language (UML). UML consists of a use case diagram, class diagram, sequence diagram, state diagram, sequence diagram and a database schema. These diagrams play a very crucial role in the analysis and designing stage of this software system’s lifecycle as these diagrams are foundational blueprints that will help me meet the functional and nonfunctional requirements during the coding/implementation stage.

A use case diagram is a way to summarize details of a system and the users within that system. The graphic depiction of the student registration system shows the interactions among different elements in the system. Use case diagrams are important because they describe the high-level functions and scope of a system.

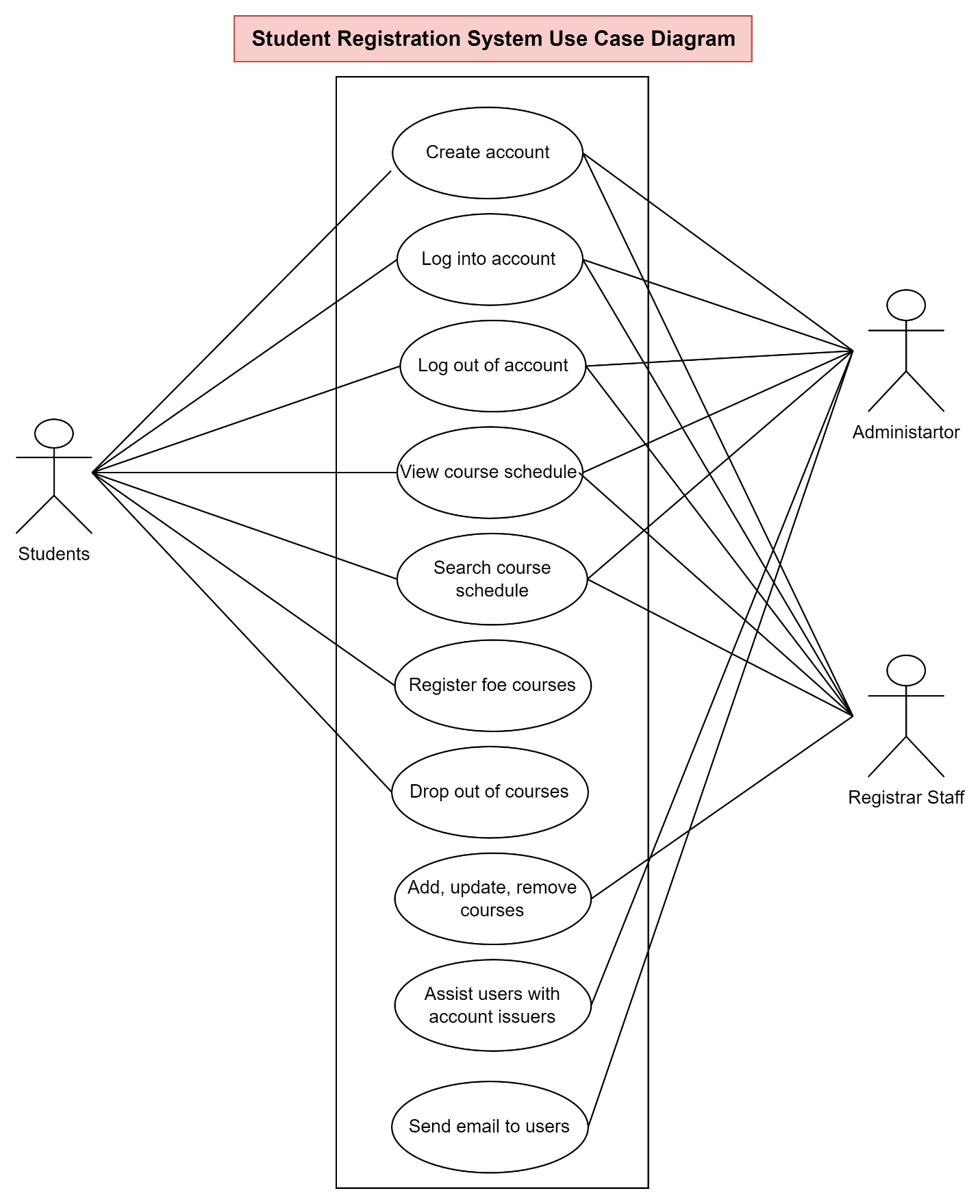


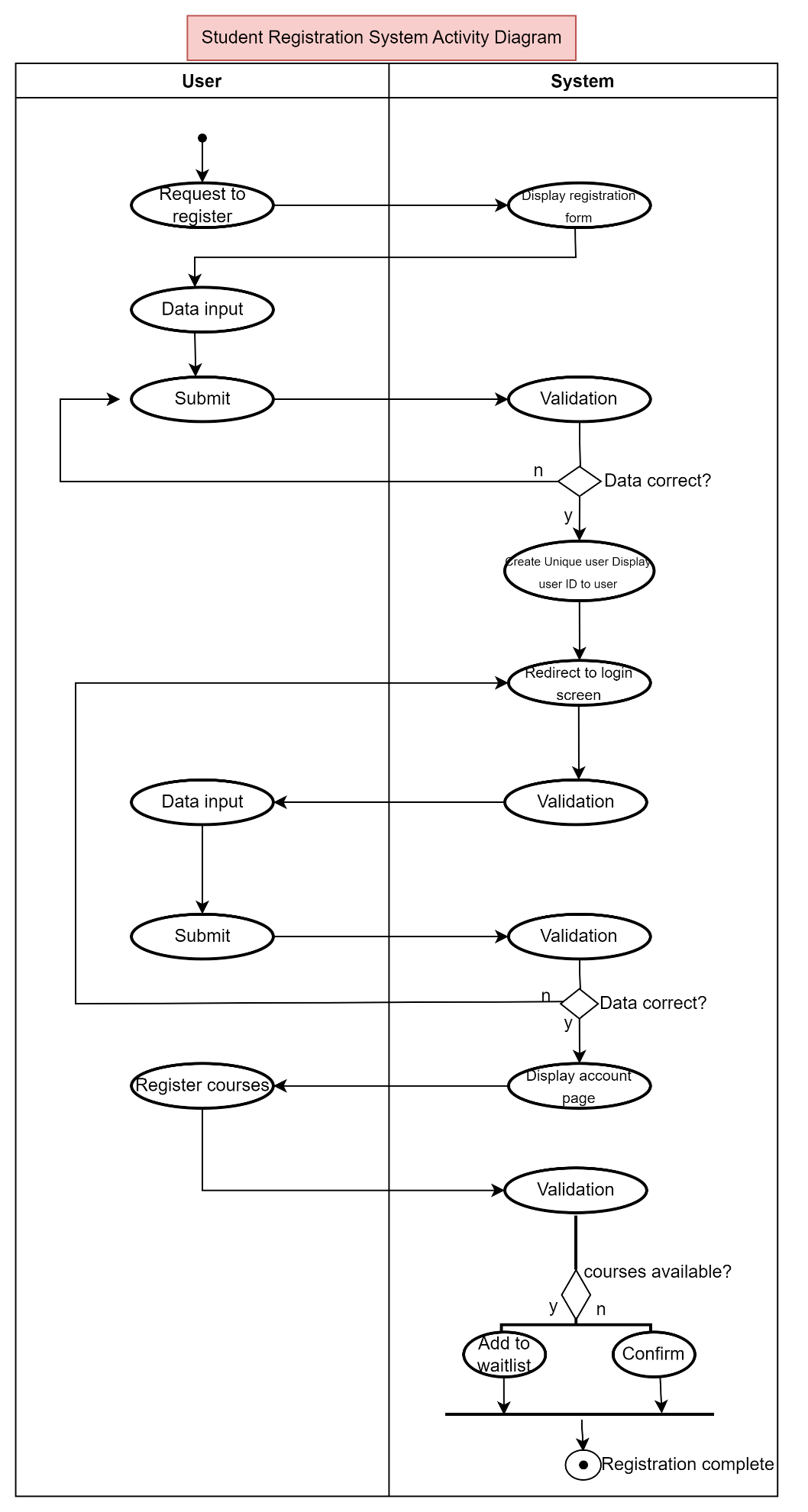
Figure-1

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system’s classes, their attributes, operations, and relationships among objects. Class diagrams are the most important kind of UML diagram and vitally important in software development because they are the best way to illustrate a system’s structure in a detailed way. This can keep things simple within the coding stage of a software system’s lifecycle. This can also save time for the database administrators as all of the classes will be defined which will promote better communication and a better flow of what is required of a software system. Figure 2 shows a class diagram of the student registration system.A screenshot of a computer

Description automatically generated

Figure-2

An activity diagram is essentially an advanced version of a flow chart that models the flow from one activity to another activity. Activity diagrams allow coordination between events that need to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single use case relate to one another.

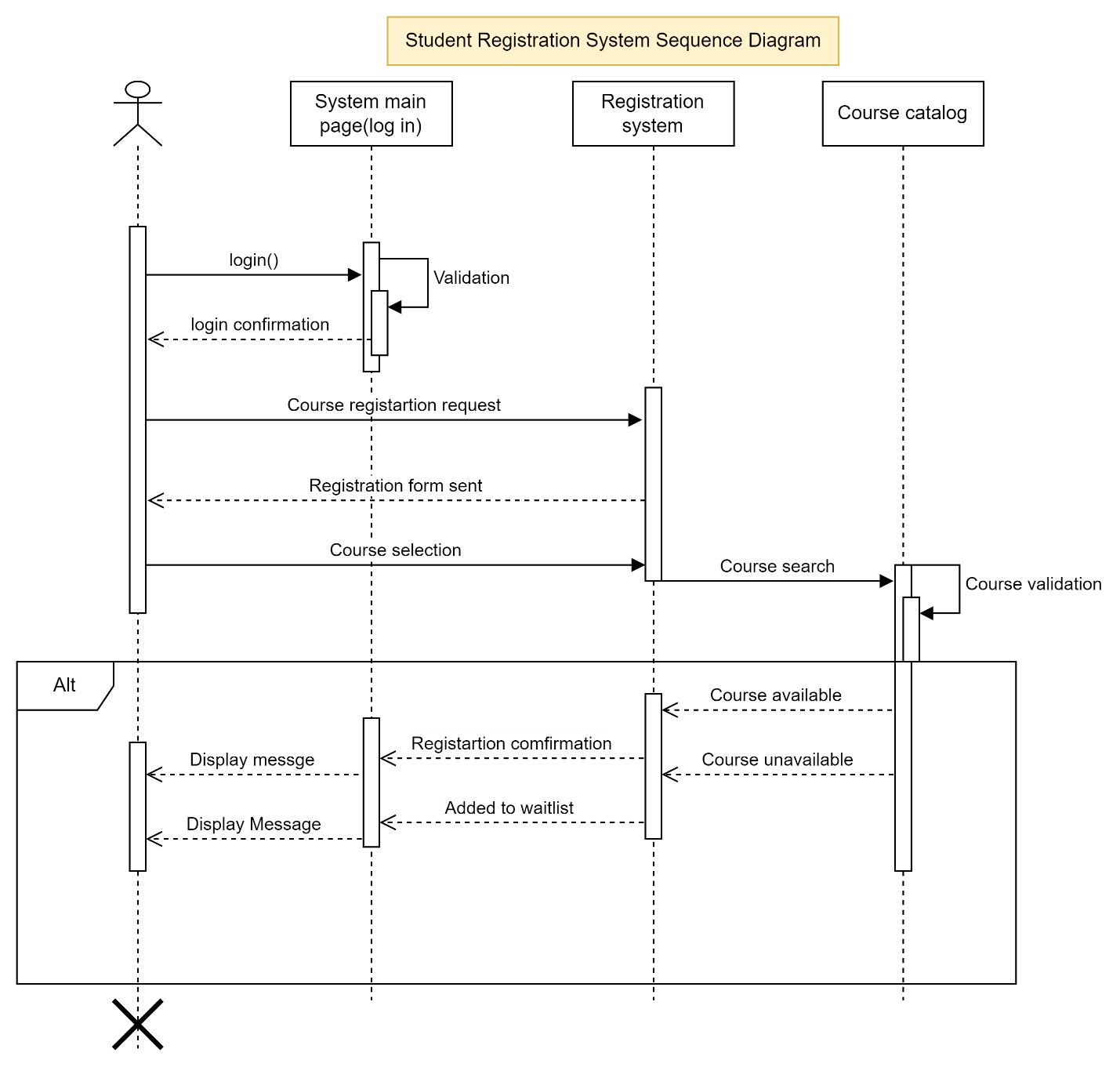
Figure-3

A state diagram is a type of diagram used to describe the behavior of a software system. State diagrams consist of states, transitions, events, and activities. State diagrams are important because they enable software development to describe the behavior of objects during their entire life cycle. Figure 4 depicts a state diagram of the student registration system.

A screenshot of a computer

Description automatically generatedFigure-4

A sequence diagram is an interaction diagram that details how operations are carried out and what messages are sent and when. Sequence diagrams show the sequence of messages passed between objects. They can also show the control structure between objects. Figure 3 shows a sequence diagram of the student registration system. Activity diagrams are important because they show the flow of one activity to another within a system or process. Figure 5 shows a depiction of the student registration system through an activity diagram.

Figure-5

A database schema is a collection of database objects, including tables, views, indexes, stored procedures, and more, that defines the structure and organization of a database. It provides a blueprint or framework for how data is stored, organized, and accessed within a database management system (DBMS). The schema defines the relationships between different entities and the rules and constraints that govern the data. Database Schemas are important because they promote data integrity, security, and data independence for software systems. Figure 6 is a depiction of a database schema for the student registration system.

A black background with white circles

Description automatically generatedFigure-6

Testing is a very important activity in every software product’s life cycle. In fact, according to  [joint.com](http://joint.com/) (2020), software testing is so important it involves almost 50 percent of a project. Software testing is defined as an activity performed for evaluating product quality, and for improving it by identifying defects and problems (Bourque & Fairley, 2004). All methods of testing require the definition of yeast criteria. These criteria are used to determine what a suitable set of test cases should be. There are many types of testing techniques and methodologies depending on the type of software product being produced and the team developing the product will determine the testing techniques that should be used. No matter the main purposes of finding defects in the software so that they can be corrected or mitigated and providing general assessment of quality (Karam & Bernal, 2018). As I stated earlier, there are many testing techniques available to software testers and these techniques are applicable to different circumstances. The techniques I will be discussing are component testing, integration testing, system testing, and acceptance testing.

A diagram of a general v model

Description automatically generatedFigure-7

       Component testing, integration testing, system testing, and acceptance testing all align with the general v-model. The main idea behind the general v-model is that development and testing tasks are corresponding activities of equal importance (Spillner & Schafer, 2012). Component testing is within the first test level. Component testing involves testing individual units or components, these components can be a function, method, or class. These different components are assigned to different actors within a software system like customers, administrators, and vendors. All components should be thoroughly tested as they ensure the correctness of use cases.

        Integration testing should be commenced under the precondition that the test subjects have already undergone the component test and have been corrected. The primary objective of integration testing is to reveal interface problems as well as conflicts between integrated parts (Spillner & Schaefer, 2012). During integration testing, different smaller components are combined to form larger units. After the components are combined, these components must be taken into account and their collaboration with other components must be examined. Once all of the components are thoroughly tested on their own, test cases will be made to evaluate the collaboration of components.

        System testing is the third test level of the general v-model. System testing checks if the integrated product meets the specified requirements (Spillner & Schaefer, 2012). System tests look at a software system from the perspective of the customer and the future user. The validation of whether the requirements are completely and appropriately implemented comes from the testers themselves. A system test is important for software systems because it allows testers to see if the system is doing what it is supposed to do as a whole.

        Acceptance testing is the final test that is conducted in the general v-model. Acceptance testing is defined as the explicit and formal testing conducted by the customers prior to officially accepting the software product and paying for it (Karam & Bernal, 2018). The acceptance test is before an actual system is deployed. The acceptance test may be the only test that the customer will be involved in because the customer needs to see the entire system from his or her perspective. The amount of testing needed for this test type depends on the product risks so, if the risks are higher, more extensive testing should be conducted. Apart from the customer performing his or her own test on the product as well. The key reason behind acceptance testing is to see how people will react to the product in the real world.

In conclusion, by constructing UML diagrams of the Student Registration System I have gained a better understanding of what is required from the system. The use case diagram displays high-level functionalities and system scope, while the class diagram offers a detailed snapshot of the system's structure. Sequence diagrams illustrate the flow of operations and message passing, while state and activity diagrams display the software system behaves. Crucially, the database schema serves as the backbone, defining the organization and relationships within the database. As the system progresses through testing phases—component testing, integration testing, system testing, and acceptance testing—it aligns seamlessly with the iterative and collaborative approach of the general V-model. Testing, constituting nearly half of the software development effort, ensures not only defect identification but also an overall assessment of quality. The culmination of these elements, from design blueprints to meticulous testing, sets the stage for a robust, user-centric, and functional Student Registration System. Creating a student registration system for an online university served as a wicked problem to me, so I conducted extensive research during the requirements, analysis, and design stages of the student registration system. I used the University of Arizona Global Campus’s website to aid me in figuring out what is actually required from a student registration system. The UML diagrams that I constructed have provided a clear blueprint for me to reference throughout the coding stage of the software lifecycle but, I feel that I will need to venture back to the SRS documentation of the software system in order to make some revisions as I undergo the coding stage.

References

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](https://uagc.instructure.com/courses/126521/modules/items/6439323) (4th ed.). Jones

& Bartlett Learning.

Guide to the software engineering - fit.edu. (n.d.). https://cs.fit.edu/~kgallagher/Schtick/Serious/SWEBOKv3.pdf