**John Grier**

**University of Arizona Global Campus**

**CST499 Capstone for Computer Software Technology**

**Instructor: Rangitsch**

**May 26, 2024**

Static modeling focuses on depicting the structure of the objects, classes, or components in the problem domain, which are specified using the keyword class, object, or component. In contrast, dynamic modeling concentrates on representing the interactions between the objects at runtime.

Figure 1-5: UML five diagrams There are mainly five diagrams in UML that we can use to model the dynamic nature of specifications. Among these diagrams, use case diagram is the name we will discuss because we have to demonstrate something: that use case diagram is dynamic. From another angle, in order to to the interaction that we have to model, we have to have internal or external factors that make this interaction. We will call them internal or external agents, namely actors.

The elements are the parts that can be connected in different ways to form a complete UML diagram. Thus, understanding the different diagrams is essential to use the knowledge in practical life systems.

Diagrams or pictures are the better solution or the best picture to describe any complex system. Some industries are using diagrams in different forms from human beings every day. We take the example of Google Maps. This is the more familiar example to clarify.

The system is better and easier to understand when we draw a UML diagram. It is impossible to use one single diagram to represent the whole system. UML provides us with several types of diagrams to cover most of the aspects of a system. Of course, depending on your requirements, you can create your own set of diagrams. Generally, diagrams are made incrementally and iteratively.

**Sequence Diagram:**

Sequence diagrams are interaction diagrams. As the name implies, they deal with sequences, which are the sequences of messages flowing from one object to another. From an implementation and execution perspective, the components of a system must interact. Using a sequence diagram, it is possible to visualize the sequence of calls within a system to accomplish a particular function.

A diagram of a graph

Description automatically generated with medium confidence

Figure 1: Sequence Diagram

**Activity Diagram:**

It is the control flow within a system that is described by an activity diagram, which contains activities as well as links. The flow can be sequential, concurrent, or branched. Activity diagrams represent a system's functions. These diagrams are prepared to illustrate the entire flow of a system. It is essential to understand the flow of controls in a system by preparing activity diagrams. This information will assist in understanding how the system will function when implemented.

A diagram of a diagram

Description automatically generated

Figure 2: Activity Diagram

**StateDiagram:**

Real-time systems are expected to be affected by internal/external events. It is these events which cause the system's state to change. In the context of a state chart diagram, a class or interface is described as changing their state as a result of an event. An internal/external factor's reaction is visualized by a state chart diagram.

A screenshot of a computer

Description automatically generated

Figure 3: State Diagram

**Use Case Diagram:**

This diagram consists of several use cases, actors, and their relationships. Essentially, it represents the use case view of a system. A use case represents a particular functionality within a system. As such, use case diagrams illustrate the relationship between functionalities and their external/internal controllers, commonly called actors.

A diagram of a user account

Description automatically generated with medium confidence

Figure 4: Use Diagram

**Class Diagram:**

Class diagrams are the most frequently utilized UML diagrams, encompassing classes, interfaces, associations, and collaborations. They provide a static representation of the object-oriented perspective of a system. Active courses within a class diagram depict system concurrency and are primarily employed for development purposes. The active class diagram is particularly prevalent during system construction.

A screenshot of a computer

Description automatically generated

Figure 5: Class Diagram

**Component testing**

In component testing, every component is individually tested separately and is not integrated with the other components, so it is regarded as a form of software testing. As we consider this type of testing from an architectural point of view, we can also refer to it as Module Testing. The software components are mainly composed of many components, making component tests, unit tests, program tests, and module tests the most commonly used types of testing. Component Level Testing is also known as this type of testing. It is one of the most widely used types of black box testing.

In such testing, the components must be in a state of independence and controllability. Each component should be easily understandable by the user. Consider a software system consisting of fifty components. During the development cycle, a tester individually tests each component before integration testing is performed. It saves time since bugs are detected at the earliest stage of the development cycle. Programmers use test structures or debugging tools to conduct this type of testing. Programmers perform component testing on the code they write with support from IDE. Any defect detected during component testing is fixed as soon as it is detected, and these defects are not maintained in maintenance. It is crucial for component testing to determine the issue. Each application component is tested during component testing to ensure it is functioning and meeting the requirements before integration testing.

**Integration testing**

QA Engineers also conduct final integration testing, an essential component of the software testing process. At this phase, they combine individual units to test them as a whole group. This type of software testing aims to identify any errors in the interaction between the individual system components.

Testing is carried out by quality assurance engineers, starting at the highest level of the project and working down to the lowest level, known as top-down integration testing. This type of testing is a flow testing approach from top to bottom, or in the context of architecture, it is known as structural testing since this method aims to test the software structure. It involves testing all the units along a structured path until all interfaces have been verified. The test process begins at the beginning and ends somewhere in between. There may be incomplete units that are not ready, but we can use stubs as substitutes or temporary replacements for these incomplete units, referred to as black-box or white-box testing.

An approach that tests bottom-up integration assumes that the tester examines units at the unit level first and then progresses upwards. This approach is practical if most units are ready at the unit level or if all of the units are ready.

Generally speaking, each approach type for integration testing has advantages and disadvantages. It depends mainly on the cost and complexity of the application. In top-down integration testing, stubs are temporary replacements for incomplete units. These stubs mimic the behavior of the actual units, allowing the testing process to continue even if specific components are not yet ready. This approach helps identify errors in the interaction between the tested components and the incomplete units, ensuring a thorough evaluation of the software structure. However, it is essential to note that the use of stubs may introduce limitations or inaccuracies in the testing process, as the behavior of the actual units may differ from that of the stubs.

**System testing**

System testing aims to verify that the software solution is functionally complete and fully integrated. This testing process ensures that the system to be tested meets the requirements specified and is ready for delivery to end users. During system testing, the entire system is checked as a whole and occurs between the integration and acceptance phases.

System Testing ensures that the entire integrated system operates according to user requirements. A QA Engineer combines all the components into a test to determine whether the integrated system meets the end user's requirements.   During system testing, several challenges may arise. One challenge is identifying and resolving any compatibility issues between different system components. Another challenge is ensuring that all the necessary data is available for testing, as it can be challenging to replicate real-world scenarios. Additionally, coordinating the testing efforts of multiple teams and stakeholders can be a challenge to ensure effective communication and collaboration.

It is considered a black box testing technique, in which the tester reviews the software without being aware of the internal structure of the code, the implementation methods, or the internal paths of it. Quality professionals use the specification at this stage to determine whether a system complies with the requirements stated in the specification.

**Acceptance testing**

Acceptance Testing (AT): This is the final stage of SWAT software testing, which is likely to be undertaken before the product is marketed to confirm that the results match the accepted acceptance criteria.

An essential principle of UML is the "acceptance criteria": they are requirements in a User Story that indicate when you have completed what meets the user's needs. Quality Assurance Engineers or developers test a software program in principle to ensure that all requirements have been met; they look at the system from the client's perspective. At this stage of development, test engineers work on the system using both white boxes (testing input and output functions). Testing engineers for software applications do more than verify transactions, as their name implies.+

As a follow-up to Alpha Testing, Beta Testing is the next step in improving the quality of your product. Beta Tests require the involvement of a group of users, whereas Black Box Testing involves a group of potential users as the end user. When browsing the internet is a tame process for evaluating a product the development team provides.

As indicated by the descriptions above, alpha and Beta Testing have different processes, strategies, and goals. It is important to note that both types of testing, Alpha and Beta, are part of acceptance testing, which helps build confidence in the final system as it prepares for launch. Success in the market is determined by having a successful product launch. Additionally, we will consider five additional dimensions of software testing that address the quality attributes of software in addition to the two core groups we examined. These factors contribute to creating a good product, including reviewing performance and stability, testing security and usability, facilitating regressions, and automating testing.

Reference:

Naiburg, E. J., & Maksimchuck, R. A. (2001). *UML for database design*. Addison-Wesley Professional.

Tsui, F., Karam, O., & Bernal, B. (2018). [Essentials of software engineering](https://uagc.instructure.com/courses/132028/modules/items/6718782) (4th ed.). Jones & Bartlett Learning.