Discussion-3 advantage and limitation of actors and use cases

Discussion Topic:

Discuss one advantage and one limitation of actors and use cases. Provide an example of each in your post. In your peer responses, share an additional example based on their posts.

My Post:

Hello Class,

In UML, use-case diagram illustrates the behavior of systems helping to capture the working requirements of those systems (IBM, 2023). In other words, the diagram describes the high-level functionality and the main scope of systems. Use-case diagrams also identify the interactions between systems use-cases and their actors. Actors and use-cases can be defined as follows:
Actors can be people, organizations, or external or internal systems that interact with the business. Use cases can be specific functionalities or services that the business provides to the actors and other use cases.
Furthermore, use cases can be divided into base use cases, which represent the core functionalities of the systems (can be modified by other use cases), and additional use cases are supplemental functionalities that can modify other use cases. Additional use cases can also be defined as supplemental functionalities that can modify other use cases.

In UML use-case diagrams, the interactions between use cases are defined as relationships that can be divided into four categories:

- Association relationships are illustrations of direct interactions between actors and use cases or between use cases.
- Include relationships are illustrations of interactions where use cases include the functionalities of another use-cases.
- Extend relationships are illustrations of interactions where use cases are extended by other use cases under certain conditions.
- Generalization relationships are illustrations of interactions where use cases are generalized versions of other use cases or actors are generalized versions of other actors. It is a parent-child relationship where the parent is the generalized version of a child.

(Helm, n.d)
The relationship between actors and use cases can be only association (direct interactions). Additionally, UML 2 does not permit direct associations between actors; however, it permits generalization relationships between actors.
These relationships define the core functionality of the UML use-case diagrams. Furthermore, the diagrams provide many advantages, making them an effective tool for capturing and representing a system's requirements. Therefore, they are very useful in the early stage of Software Development (SD).

Advantages

Actors and use cases provide a user-centric approach when modeling a system's requirements. This helps ensure that the correct system is developed by capturing the requirements from a user perspective (Firesmith, n.d.). For example, in a Hospital Management System (HMS) case-use diagram, a patient can be identified as an actor "A10-Patient" and a use case that illustrates the ability to book an appointment with a doctor "UC30-BooksConsultation." By connecting "A10-Patient" and "UC30-BooksConsultation" with an association relationship capturing a requirement directly from the patient's point of view (the need to schedule a doctor appointment) and with a specific system functionality (use-case) designed to meet that user need (example from Unhelkar, 2018). This demonstrates the actors' and use cases' user-centric approach to modeling requirements.

In addition to providing a user-centric approach they also provide the following advantages:

- Use cases and actors are easy to recognize and their descriptions are written in natural language making it easy to understand and providing an excellent way to communicate with customers and users (Firesmith, n.d.).

The following list is from “Chapter 6 – Use Case Models-1: Actors and Use Cases. Software engineering with UML” (Unhelkar, 2018 b, p. 92):

- Use cases help the business analyst to document requirements in a commonly accepted format in the problem space of the project.
- The actor, through the use cases, specifies the suite of interactions with the system.
- Use cases capture the functional aspects of the system. More specifically, they capture the business processes carried out in the system. They are usually developed by domain experts and business analysts, resulting in the effective documentation of functionalities.
- Since use cases document the complete functionality of a system, no separate functional requirements document is needed (although additional operational and interface requirements or additional details such as the referenced material may be available or placed in a separate document).
- Use cases facilitate tracing of requirements. By providing well-organized documentation on the requirements, a use case provides a trace for a particular requirement throughout the system. This is especially helpful in creating and executing acceptance tests by users.
- Use cases can help in the creation of prototypes. Developers can select a use case and produce a proof-of-concept prototype of the system that will validate system requirements.
- Documentation of a use case provides a means for creating activity diagrams. The documentation of the flow within the use case can also be influenced and improved by the activity diagram(s) drawn for a use case.
- Specifications and documentation of use cases also provide a rich source of information for the identification of business entities. These business entities can be put together in a suite of class diagrams—providing vital information in the model of the problem space.
- Use cases can also provide a starting point for sequence diagrams—based on the scenarios (or instances of behavior) documented within a use case.
- Use cases are the basis for test case development.
- Use cases aid in requirement mapping: matching a requirement to a software feature to the approved test case.

Limitations

However, UML use cases also have some limitations. By primarily focusing on a system's requirements, they may not capture non-functional requirements. In other words, while they capture well what the system should do, they do not demonstrate well how the system should do it. Non-functional requirements, such as performance, security, and usability are often not documented in use cases. For instance, using the HMS example, a use case "UC10-RegistersPatient" would describe well the process of registering a new patient; however, it would not explicitly mention non-functional requirements such as performance (how quickly should the registration process be completed?) or security (how should patient data be protected during registration and storage?)

In addition to not capturing non-functional requirements properly, UML use cases experience the following limitations:

The following list is from "Chapter 6 – Use Case Models-1: Actors and Use Cases. Software engineering with UML" (Unhelkar, 2018 b, p. 92):

- Use case documentation is not standardized. This leads to confusion and debates on what makes up a good use case. Most projects proceed on the basis of a template (see previous discussion).
- Use cases are not object-oriented in nature. Therefore, they are not an ideal mechanism to model design-level constructs in the solution space (where object orientation plays an important role).
- Use cases do not have a granularity standard. Therefore, sometimes use cases are written as huge descriptive documents, resulting in the inability of modelers to capitalize on the reusable and organizational aspect of use case modeling. Alternatively, too brief a description will result in a large number of miniscule use cases—making them less comprehensible and manageable.
- Use cases do not provide a good basis for coding. Their documentation provides a foundation for subsequent modeling, but not for code generation.

To summarize, UML use-case diagrams, with their actors, use cases, and relationships, provide a user-centric approach to capturing a system's requirements which is essential in the early stage of Software Development. They excel at illustrating and capturing the "what" a system should do. However, they are not efficient at capturing and illustrating the "How" these requirements should be implemented. More specifically, they struggle to capture non-functional requirements, and they do not provide a good basis for coding generation.

-Alex

References:

Firesmith D., G. (n.d.). *Use Cases: the pros and cons*. Knowledge Systems Corporation. https://www.cs.hmc.edu/~mike/courses/mike121/readings/reqsModeling/firesmith.htm

Helm, J. (n.d.). Business use-case model. *Rational unified process*. Fall 2023 SWEN 5135 Configuration Management course. University of Houston at Clear Lake. https://sceweb.uhcl.edu/helm/RUP_Folder/RationalUnifiedProcess/process/modguide/md_bucm.htm

IBM documentation (2023, September 21). Use-case diagrams. *IBM Rational Software Architect documentation*. IBM. https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=diagrams-use-case

Unhelkar, B. (2018 a). Chapter 6 – Use case models-2: Use case diagrams and requirements modeling. *Software engineering with UML*. CRC Press. ISBN 9781138297432

Unhelkar, B. (2018 b). Chapter 5 – Use case models-1: Actors and use cases. *Software engineering with UML*. CRC Press. ISBN 9781138297432