COMP 3710 Wireless Software Engineering
Spring 2015

Essential Issue

How do you convey your understanding of a client's needs?

Overview

The act of determining client needs is referred to as "requirements analysis." It is the most difficult, and, perhaps, the least emphasized topic in today's college curricula because it demands skills that can only be obtained by extensive exposure to a variety of real clients. Doing it well requires a unique blend of people and technical skills. An analyst has have a basis in the domain of the client; has to understand the nuances of trust and communication; has to walk a thin line between underselling and overselling capabilities; and has to be able to articulate requirements such that they can be validated by the client and understood by the designer.

We lack the time to explore specific techniques for conducting a requirements analysis, but we will examine the artifacts produced by the requirements analysis process in the hope that you can extrapolate the steps you might take to understand client needs. Specifically, we will be employing use cases, system sequence diagrams, and domain models.

Use cases are a typical starting point for requirements analysis because they give us an idea of how the proposed solution is going to be used. We can then delve deeper into what our software does by treating the software as a *black box* (i.e., something to which we supply input and receive output without knowing its internals) and defining its interaction with entities outside the software boundary. We do this by describing, from the users' perspective, the behavior of our black box and the elements that our black box handles. The system sequence diagram (SSD) -- a special case of UML sequence diagram -- depicts interaction and the domain model (DM) -- a special case of a UML class diagram -- depicts what our system manipulates.

un

Guiding Questions

1. What are the four fundamental characteristics of modeling that a software engineer needs to keep in mind? How is each supported by the use case diagram and accompanying use case descriptions?
2. What is the purpose of a use case? What are the strengths of use cases? Weaknesses?
3. What software engineering role do use cases play? What software engineering activity do they best support?
4. What is the difference between a use case diagram and a use case? What is the purpose of each?
5. What is the purpose of identifying the "boundary" of a system when developing a use case description?
6. What are use case actors? How do you go about identifying an actor? How much control does your software have over what an actor does? Should a use case have a communication line from one actor directly to another?
7. What heuristics determine whether a use case is "good"?
8. What are coupling and cohesion in the context of use cases?
9. What does it mean to "write a use case at the 25-cent level"?

10. What stereotypes do use cases typically employ?
11. What is the meanings of the <<include>> annotation on a use case diagram?
12. What forms can the use case text take?
13. What information goes into a use case (also referred to as a use case table)? Why?
14. What is a UML interaction diagram? What does a sequence diagram model accomplish?
15. What is a system sequence diagram (SSD)? What does it describe? What constitutes a "good" diagram? What differentiates a sequence diagram from a system sequence diagram?
16. How do you describe the order in which interactions take place in an SSD?
17. What is an interaction frame and what purpose does it serve? What is the purpose of the common frames: loop, alt, opt, par, critical, ref?
18. How do you describe interactions that are independent of each other?
19. What is a UML structure diagram? What is the purpose of a class diagram?
20. What is a domain model (DM)? What subset of a class diagram does it use? What constitutes a "good" domain model?
21. Describe how you go about identifying the items that should go in a DM?
22. How do you model "things" in a DM? Attributes? Relationships among things?
23. What is the difference between a "thing" and an attribute? How do you distinguish the difference?
24. Why don't we describe operations in a DM?
25. What decorations can go on a relationship line?
26. How do the SSD and DM tie back to the use cases?

Resources

Here are some resources that might be of assistance:

1. COMP3710 Canvas -> Pages -> Software Engineering Resources
2. http://www.uml.org/

Exploration Activity

Develop a use case diagram, use cases, system sequence diagram(s), and a domain model for the Milton Bradley game, *Simon*. "The game unit has four large buttons, one each of the colors red, blue, green, and yellow. The unit lights these buttons in a sequence, playing a tone for each button; the player must press the buttons in the same sequence. The sequence begins with a single button chosen randomly, and adds another randomly-chosen button to the end of the sequence each time the player follows it successfully. Gameplay ends when the player makes a mistake or when the player wins (by matching the pattern for a predetermined number of tones)" [Wikipedia].

Suggestions:

1. What is the boundary of your term project? In other words, if you were to draw a box representing your software, what would be outside the box that provides input to and receives output from your software?
2. Who uses your software? Consider active actors that initiate actions as well as passive actors that are the recipients of actions.
3. How does each actor put your software to use? For complex uses (or, if you have to use "and" to describe use), consider breaking the use into smaller parts.
4. Consolidate the boundary, actors, and uses into a *use case* diagram.
5. For each use case in your diagram, describe what actions that are observable by the actor associated with the use case.
6. Examine your use case diagram and individual use cases to ensure the information is complete, depicts the appropriate level of detail, and has no implementation details. Revisit the boundary of your system from the previous challenge.
7. For each actor in your system, identify what interaction takes place between the actor and *your system as a black box*. Do not show anything inside the system boundary … you'll figure this out later. If you are stuck, consider writing a paragraph describing your software. The verbs and verb phrases are likely to describe interactions.
8. Draw a system sequence diagram showing what input each actor gives to your software, what your software returns to the actor, and in what order this interaction takes place.
9. For each actor in your system, identify the "things" in your software that your user needs to know about in order to understand the software's functionality. Make sure you don't describe implementation-specific things, only items that are independent of the software. If you are stuck, revisit your descriptive paragraph. The nouns and noun phrases are likely to describe "things." You'll need to choose nouns and noun phrases that name domain-level things (meaning, things that are independent of the software implementation) and ignore nouns and noun phrases that related to how your software is implemented in software.
10. Draw a class diagram (called a domain model) showing "things" as boxes. Include attributes of "things" (such as color, size, position, etc.) in the boxes as well. Describe relationships between "things" as lines. Make sure to annotate the lines with the decorations that describe the various relationships (peer association, is-a, part-of, aggregation-of). Don't worry about the operations of the "things" at this point; you'll work on that later.

Submission	No submission