Use Cases: The Technique

From the Rational Unified Process
Requirements Workflow Detail: Analyze the Problem

Analyze the Problem

Define the System
Manage the Scope of the System
Refine the System Definition
[Can't do all the work]
[Work in scope]
[Addressing correct problem]
[Existing System]
[New Input]
[New System]
Manage Changing Requirements

Find Actors and Use Cases

Capture a Common Vocabulary
Develop Requirements Management Plan
Develop Vision
Use Case Model (actors only)
Stakeholder Requests
Vision

Iteration Plan
Vision
Glossary
Software Development Plan
Requirements Management Plan
Customer
System Analyst
End User
Other Stakeholder

Business Rule

SE 555 Software Requirements & Specification
Purpose of “Analyze the Problem”
Workflow Detail

- Elicit and collect stakeholder needs and requests
  - “Wish List” is input to high-level features in Vision Document
  - Drives requirements in use cases, use-case model, and supplementary specifications

- Result
  - Refined Vision with prioritized features and their critical attributes (such as, benefit, rationale, level of effort to implement, risk, stability, trace to stakeholder source, etc.)
  - Initial identification and brief description of actors and use cases
  - Non-functional requirements (in Supplementary Specifications)
  - Updated Glossary
Activity: Find Actors and Use-Cases

Purpose
- Outline the functionality of the system
- Define the scope of the system
- Define who and what will interact with the system
- Create diagrams of the use-case model

Steps
- Find Actors
- Find Use Cases
- Describe How Actors and Use Cases Interact
- Present the Use-Case Model in Use-Case Diagrams
- Develop a Survey of the Use-Case Model
- Evaluate Your Results
Step: Find Actors

- Ask
  - Which user groups require help from the system to perform their tasks?
  - Which user groups are needed to execute the system's most obvious main functions?
  - Which user groups are required to perform secondary functions, such as system maintenance and administration?
  - Will the system interact with any external hardware or software system?
- Name the actor to clearly describe the actor’s role
- Briefly describe the actor’s area of responsibility and what the actor needs the system for
Step: Find Use Cases

- Consider what each actor requires of the system
  - What are the primary tasks the actor wants the system to perform?
  - Will the actor create, store, change, remove, or read data in the system?
  - Will the actor need to inform the system about sudden, external changes?
  - Does the actor need to be informed about certain occurrences in the system?
  - Will the actor perform a system start-up or shutdown?
  - Will the actor support and maintain the system?

- Name the use case

- Briefly describe the use case, clarifying its purpose

- Outline the basic and alternative flow of events

- Collect additional requirements as supplementary specifications

- Iteratively add, remove, combine, and divide the use cases
Step: Describe How Actors and Use Cases Interact

- Establish which actors will interact with each use case
- Define a communicates-association that is navigable in the same direction as the signal transmission between the actor and the use case
  - Signal transmissions
    - Let the communicates-associations be navigable in both directions denoted by a line
    - Optional convention: use arrowhead to show whether the actor initiates the interaction or the system initiates the interaction
- Define, at the most, one communicates-association for each actor-and-use-case pair
Step: Develop a Survey of the Use-Case Model

- Write a Survey Description that includes
  - Which are the primary use cases of the system (the reason the system is built)
  - Typical sequences in which the use cases are employed by the users
  - Relationships between use cases (generalizes, extends, includes relations)
  - System delimitations – things that the system is not supposed to do
  - Important technical facts about the system
  - The system's environment, for example, target platforms and existing software
  - Specify functionality not handled by the use-case model
Step: Evaluate Use-Case Model

- All necessary use cases are identified.
  - Verify that the use-case model addresses all functional requirements
- Any unnecessary use cases are identified
  - Use cases that provide little or no value
  - Use cases that should be part of larger use cases that do provide value
- If the behavior of each use case is performed in the right order
- If each use case's flow of events is as correct, complete, and understandable as it could be at this stage
- If the survey description of the use-case model makes it understandable
Requirements Workflow Detail: Define the System

Define the System
Purpose of “Define the System” Workflow Detail

- This activity is a refinement of the results of the “Analyze the Problem” workflow, emphasizing:
  - Structuring the requirements
  - Clarifying the scope
  - Building consensus
- Align the project team in their understanding of the system
- Perform a high-level analysis on the results of collected stakeholder requests
- Refine the Vision to capture the key features that characterize the system
- Refine the use-case model to include outlined use cases
- Begin to capture the results of the requirements elicitation activities in a more structured manner
Requirements Workflow Detail: Refine the System Definition
Activity: Detail a Use Case

- **Purpose**
  - To describe one or more of the use case's flow of events in sufficient detail to enable sw dev to begin
  - To describe the use case specification to the understanding & satisfaction of the customers & users

- **Steps**
  - Detail & structure the Flow of Events of the Use Case
  - Describe the Special Requirements of the Use Case
  - Describe Communication Protocols
  - Describe Pre & Postconditions of the Use Case
  - Describe Extension Points, <optional>
  - Evaluate Your Results
Step: Detail the Flow of Events of a Use Case

- “Find Actors and Use Cases” activity: Use this as a starting point, and gradually make it more detailed
- Decide on the following points before describing the use cases so that you are consistent across use cases
  - How does the use case start?
    - “The use case can start when … happens.”
  - How does the use case terminate?
    - “When … happens, the use case terminates.”
  - How does the use case interact with actors?
    - Say exactly what will reside inside the system, and what will reside outside the system
    - Structure the description as a series of paragraphs
    - Each paragraph expresses an action in the format: “When the actor does … , the system does ….”
Step: Detail the Flow of Events of a Use Case

- Decide on the following points before describing the use cases so that you are consistent across use cases (cont’d)
  - How does the use case exchange data with an actor?
    - For example, “The use case starts when the User logs into the system by giving his name and password”
  - How does the use case repeat some behavior?
    - Try to express this in natural language; avoid code-like constructs
  - Are there any optional situations in a use case's flow of events?
    - “The actor chooses one of the following, one or more times: a), b), c), …”
  - How should the use case be described so that the customer and the users can understand it?
    - Use the glossary and other domain terminology
Content of a Use-Case Description

- How and when the use case starts
- When the use case interacts with the actors, and what data they exchange
  - Note: You must be explicit and complete regarding the data exchanged between the actors and the use case
  - Avoid vague terminology such as "for example", "etc. " and "information"
- How and when the use case uses data stored in the system, or stores data in the system
- How and when the use case ends
- Describe alternate, odd or exceptional flows of events
Other "do's and don'ts" to consider when you describe a use case

- Describe the flow of events, not just the use case's functionality or purpose.
- Describe only flows that belong to the use case, not what is going on in other use cases that work in parallel with it.
- Do not mention actors who do not communicate with the use case in question.
- Do not provide irrelevant detail when you describe the use case's interaction with any actor.
- Do not describe the details of the user interface, unless it is necessary to understand the behavior of the system.
- If the order of the subflows described for the use case does not have to be fixed, do not describe it as if it is.
- Use the terms in the common glossary
- Consider the following in writing the text:
  - Use straightforward vocabulary.
  - Write short, concise sentences.
  - Avoid adverbs, such as very, more, and rather.
  - Avoid compound sentences
Step: Structure the Flow of Events of the Use Case

- A use case's flow of events can be divided into several subflows when:
  - The use case can proceed from one of several possible paths, depending on the input from a given actor, or the values of some attribute or object
  - The use case can perform some subflows in optional sequences
  - The use case can perform several subflows at the same time
- You must describe all optional or alternative flows
  - It is recommended that you describe each subflow in a separate supplement to the Flow of Events section
- You can illustrate the structure of the flow of events with an activity diagram.
Some Guidelines to Structure the Flow of Events

- The alternative flows of events cover behavior of optional or exceptional behavior, and variations of the normal behavior
  - Think of the alternative flows of events as "detours" from the basic flow of events, some of which will return to the basic flow of events and some of which will end the execution of the use case
- A subflow should be a segment of behavior within the use case that has a clear purpose, and is "binary" in the sense that you do either all or none of the actions described
Some Guidelines to Structure the Flow of Events

- You need to describe the following for each "detour" to the basic flow of events:
  - Where in the basic flow of events the alternative behavior can be inserted
  - The condition that needs to be fulfilled for the alternative behavior to start
  - How and where the basic flow of events is resumed, or how the use case ends
Step: Describe the Special Requirements of the Use Case

- Describe any requirements that can be related to the use case, but that are not taken into consideration in the Flow of Events of the use case
  - Described in the Special Requirements of the use case
  - Such requirements are likely to be nonfunctional
Step: Describe Communication Protocols

- Develop a communication protocol if the actor is another system or external hardware.
- The description of the use case should state if some existing protocol (maybe even a standardized one) is to be used.
- If the protocol is new, you must fully describe it during object-model development.
Preconditions and Postconditions

- A precondition is the state of the system that is required before the use case can be started.
- A postcondition is the state the system is in after the use case has ended.
- Postconditions can be a useful tool to describe use cases:
  - First define what the use case is supposed to achieve, the postcondition.
  - Then describe how to reach this condition (the flow of events needed).
Step: Describe Preconditions of the Use Case

- A precondition on a use case explains the state the system must be in order for the use case to be possible to start.
- The states described by pre- or postconditions should be states that the user can observe.
- Take care to describe the system state.
  - Avoid describing the detail of other incidental activities that may have taken place prior to this use case.
- A precondition is a constraint on when a use case can start.
  - It is not the event that starts the use case.
- A precondition for a use case is not a precondition for only one subflow, but you can define preconditions and postconditions at the subflow level.
Step: Describe Postconditions of the Use Case

- A postcondition lists possible states the system can be in at the end of the use case
  - It is also used to state actions the system performs at the end of the use case,
- A postcondition should be true regardless of which alternative flows were executed
  - If something could fail, it is covered in the postcondition by saying "The action is completed, or if something failed, the action is not performed", rather than just "The action is completed"
  - Subflow-specific postconditions can be defined at the subflow level
- When you use postconditions together with extend-relationships, be careful that the extending use case does not introduce a subflow that violates the base postcondition.
Do Not Sequence Use Cases with Pre/Postconditions

- You should avoid using pre- and postconditions to create a sequence of use cases
  - There should never be a case where you have to first perform one use case, then another, in order to have a meaningful flow of events
  - Exception: when a common “sub-use-case” is factored out
    - For example, a “LogIn” use case
- If you feel a need to do this, the sequentially dependent use cases should be combined into a single use case
Step: Describe Extension Points

- An extension point opens up the use case to the possibility of an extension

- If the use case is to be extended by another use case, you need to describe what the extension points are
  - Extension point name
  - A list of references to one or more locations within the flow of events of the use case

- Using named extension points will help you separate the specification of the behavior of the extending use case from the internal details of the base use case
Structuring the Use-Case Model

Purpose

- To extract behavior in use cases that need to be considered as abstract use cases
  - Examples: common behavior, optional behavior, exceptional behavior, and behavior that is to be developed in later iterations
- To find new abstract actors that define roles that are shared by several actors

Steps

- Establish Include-Relationships Between Use Cases
- Establish Extend-Relationships Between Use Cases
- Establish Generalizations Between Use Cases
- Establish Generalizations Between Actors
- Evaluate Your Results
Establish Include-Relationships Between Use Cases

- If a use case contains a segment of behavior where the result, not the method for getting the result, is of importance to the rest of the use case, this behavior can be factored out to a new inclusion use case.
- Inclusion describes a behavior segment that is inserted into a use-case instance that is executing the base use case.
- When describing the flow of events of the base use case, you should refer to the inclusion at the location in which the inclusion is inserted.
Include-Relationship Between Use Cases

- Use the include-relationship to:
  - Factor out behavior from the base use case that is not necessary for the understanding of the primary purpose of the use case, only the result of it
  - Factor out behavior common to two or more use cases
- The base use case has control of the relationship and can depend on the result of performing the inclusion
  - Only the base use case knows of the relationship between the two use cases
  - No inclusion use case knows what other use cases includes it
- A communication-association to an actor (line) is only needed if the behavior in the inclusion explicitly involves interaction with an actor
Establish Extend-Relationships Between Use Cases

- If a use case has segments of behavior that are optional or are exceptional in character, and that do not add to the understanding of the primary purpose of the use case, factor those out to a new extension use case.
- In the base use case you declare extension points, which define where in the base use case extensions may be made.
Extend-Relationships Between Use Cases

- Use the extensions to:
  - Show that a part of a use case is optional, or potentially optional, system behavior
    - Separate optional behavior from mandatory behavior
  - To show that a subflow is executed only under certain (sometimes exceptional) conditions
- Only the extension use case knows of the relationship between the two use cases
  - The base use case only knows it has extension points, it doesn't know what extension use cases are using them
Establish Generalizations Between Use Cases

- If two or more use cases have similarities in structure, behavior, and purpose, you can factor out the common behavior to create a new parent use case.
  - The original use cases will then be child use cases in generalization-relationships with the parent.
  - The child use case inherits all behavior described for the parent use case.
  - When a use-case instance follows the description of a child use case, it also needs to follow the description of the parent use case to be considered complete.
Generalizations Between Use Cases

- Generalization is used when you find two or more use cases that have commonalities in behavior, structure, and purpose
  - When this happens, you can describe the shared parts in a new, often abstract, use case, that is then specialized by child use cases
- In the flow of events of the child use case you need to explain how the child will modify the inherited behavior sequences by inserting new segments of behavior
- Only the child use case knows of the relationship between the two use cases
Differences Between Include and Generalization

- With use-case-generalization, the execution of the children is dependent on the structure and behavior of the parent (the reused part)
  - In an include-relationship the execution of the base use case depends only on the result of the function that the inclusion use case (the reused part) performs
- In a generalization the children share similarities in purpose and structure
  - In the include-relationship the base use cases that are reusing the same inclusion can have completely different purposes, but they need the same function to be performed
Establish Generalizations Between Actors

- Several actors can play the same role in a particular use case.
- Actors with common characteristics should be modeled by using actor-generalizations.
- A user can play several roles in relation to the system (the user corresponds to several actors):
  - Represent the user by one actor who inherits several actors.
  - Each inherited actor represents one of the user's roles relative to the system.

A Teller and an Accountant, both of whom check the balance of an account, are seen as the same external entity by the use case that does the checking. The shared role is modeled as an actor, Balance Supervisor, inherited by the two original actors.
Step: Evaluate your Results

- Review and discuss the use case with the stakeholders, so that they have a clear understanding of the use case and agree on its description.
- The use-case description is complete only when it describes everything the use case performs, implements, or otherwise allows from beginning to end.
- Check that the use case exhibits the properties that characterize it as a "good" use case.
Review Requirements

- **Purpose:** To formally verify that the results of Requirements conform to the customer's view of the system

- **Participants**
  - Reviewer
  - Analyst
  - Stakeholders - customers and end-users (where possible)
  - Change Control Manager (when reviewing change requests)
  - Designer (optional)
  - Project Manager (optional, usually at phase start)