Use Case Modeling

There are several ways to capture and organize software system requirements. The first is to use a series of functionally decomposed statements similar to the following:

> "The system shall permit the user to access customer account information based on security settings"

The requirement statements are then organized into a hierarchy of such statements that are grouped by some common functional theme, such as billing or account management. This approach can be termed, Structured Requirements, since all of the statements are independent except for the overall placement into the requirement hierarchy.

This approach, while common, has the significant drawback that there is little or no direct linkage between one requirement statement and the next. This forces the developer and tester to interpret this relationship, with the result that their guess is often incorrect. Moreover, project managers cannot use this requirement structure to plan or track progress, since there is little to guide the PM in terms of complexity or effort.

The second major technique is to create a series of system behavior flows, known as system use cases (or "user stories" in eXtreme Programming). This technique was pioneered in the early 1970's by Ivar Jacobson [1], and represents a very different approach to the problem of capturing and organizing system requirement information. In this approach, the business work flows are captured into a set of well defined Use Cases (e.g. Apply Payment, Manage Customer Account, Measure Drill-Head Pressure), that are composed of one or more Scenarios; a basic flow and one or more alternate and exceptional flows. In this way the Structured Requirement statements are organized into a sequence that eliminates the need for developers to guess at the system behavior. Testers can use the Scenarios in developing test cases, since each scenario has a defined start and end point. Finally, Program Managers can apply Use Cases to organize and track project progress by measuring the number of Scenario and Use Cases complete and tested [2].

This tutorial introduces the basic technique of Use Case based requirements capture. It discusses use case and scenario identification, definition of flows, and diagram presentation of Use Case relationships.

Use Case Modeling

The first step in Use Case based requirements modeling is to review the Stakeholder Request artifact (features list). This will form the basis for discovery of use cases, and provides guidance in organizing system functionality into Use Case named categories.

Use Cases are subdivided into Scenarios, where the behavior of the use case is fully defined. There are three kinds of Scenarios that may occur in a Use Case:

- **Basic** – the Basic Scenario is the *most common* flow rather than necessarily the first logical business flow. For example, View Customer Account would be the

Basic Flow, even thought Create Account would need to be performed first to establish the customer account.

- **Alternate** (and SubAlternate) – the Alternate Scenario flows may branch from the Basic Flow (and possibly rejoin at a later point, this is called *reentrant*), or they may be independent of the Basic Flow[1]. Alternate flows are considered to be less common than the Basic Flow, but still expected during normal business operations. SubAlternate flows may be used where a Scenario is very complex, with multiple branch points. It is possible to have Sub-SubAlternate flows, but this is usually and indication that the Use Case is too complex and needs to be divided.

- **Exceptional** – The Exceptional Scenarios are where error or other unexpected conditions occur during the Basic or Alternate flows. These errors may include invalid data entry, loss of connectivity to other necessary networked systems, transaction failure, or other user-facing error conditions. Exceptional paths are not intended to capture low level code based error handling (e.g. null pointer, try/catch exceptions, etc.).

The following general steps are followed to create the initial use case model:

1. Number each Stakeholder Request and list on a whiteboard (or flipchart tacked to a wall).
2. Begin with a use case named for each major grouping of requests (e.g. Manage Customer Orders).
3. Assign Actors to interact with use cases based on the Problem Statement
4. Note dependencies between use cases (e.g. «include» relations).
5. Refine requests by combining into Scenarios. Each Scenario must have a defined entry point and at least one end point. SubScenarios may be used to break-down complex flows.
6. Create UML Activity Diagrams or flow-charts to illustrate the flow for each major scenario (Basic, Alternate, and Exceptional).
7. Refine use cases by incorporating and linking Scenarios.
8. Ensure all Requests map to one or more use cases.
9. Ensure all Actors are associated to one or more use cases

➥ NOTE: During the modeling effort a Glossary should be created if one does note exists from earlier efforts.

Determining dependencies between use cases is a valuable way to determine development priorities. A use case that is required by several others (e.g. «include») is likely to be considered a high-risk use case because of dependencies, and should be developed as early in the project as possible.

[1] If there are many independent Alternate flows, consider creating a new Use Case for them since they are likely to be unrelated to the current Use Case.

Use Case/Scenario Identification

Identification of use cases requires practice and patience. Typically, use cases take the form of «verb» - «adjective» - «noun object», such as Record Spark Firing Pattern, or Monitor Brake Pressure (see Figure 1). The name of a use case should be as informative as possible, while avoiding being too specific. For example, Add Customer Record, is a poor choice since the related operations of Modify Customer Record, Delete Customer Record, and Review Customer Record would all require separate, single flow use cases. Instead the name Manage Customer Account, would encompass all of these operations (which can now be assigned to Scenarios), and permit additional related dependencies to be directly indicated (e.g. Security Access).

Identifying and correctly determining the level of abstraction for a use case model and associated Scenarios requires several iterations of creation, revision, and review (see Set A, Part 4. Abstraction and Leveling). However, there are several rules of thumb to guide the use case modeler:

- If a use case basic flow contains less than 3-5 activities, consider combination with another related use case or scenario
- If a use case contains more than 5 scenarios consider splitting into several use cases
- Use case names should be short, declarative verb-object constructs (e.g. 'Prepare Reports', 'Monitor Transactions')
- Combine CRUD operations (create, retrieve, update, delete) into a single use case as separate scenarios (e.g. 'Maintain Use Profile')
- Combine cohesive (strongly-related) functionality into on or two tightly coupled use cases
- Factor out common behavior from several related use cases, see Figure 1 and 2

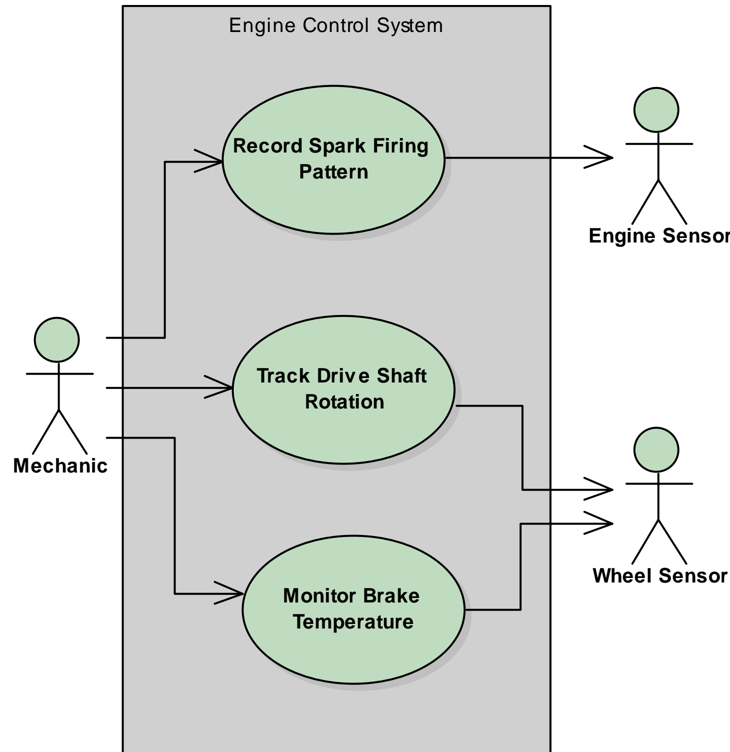


Figure 1. Use Case Example - Engine Control System

Assignment to Stakeholder Requests

One or more Stakeholder Requests should trace to each use case in the model. This permits the analyst to track progress on defining Stakeholder Requests, and also reduces the chance of a critical omission.

Example:

Engine Control System - Stakeholder Requests
1. Then Engine tuning sensors are to be monitored every 0.1 seconds 2. All reported sensor values shall be stored for a period of 24 hours 3. The main drive shaft rotation rate and operating time shall be recorded 4. Maintenance personnel shall be able to query the system for any sensor reading 5. The temperature of the main braking system will be monitored every 0.5 seconds 6. The primary spark firing pattern will be automatically calibrated to factory settings when anomalies are detected.

Use Case Mapping (based on Figure 1)
Monitor Brake Temperature – 1, 2, 4, 5
Track Drive Shaft Rotation – 1, 2, 3, 4
Record Spark Pattern – 1, 2, 4, 6

Notice that all of the use cases support Stakeholder Requests 1, 2 and 4. In the revised use case model the Retrieve Sensor Readings can be created to encapsulate Request 1 and 2. Request 4 is a general retrieval request and will likely be treated as a user interface.

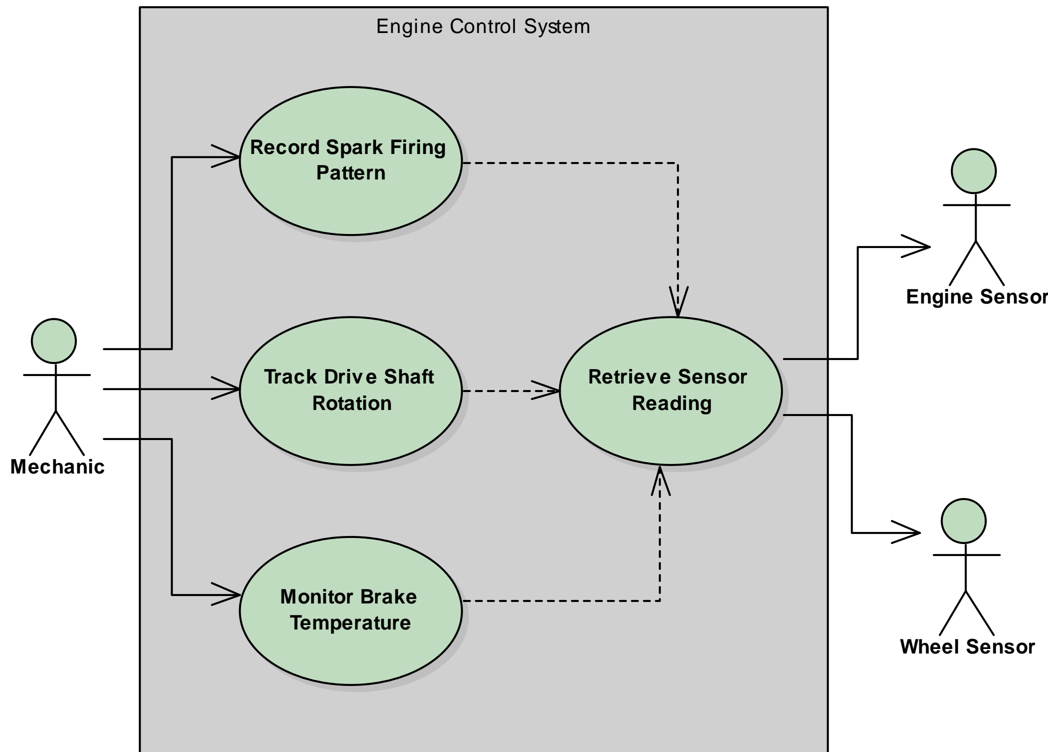


Figure 2. Use Case Example - Refactored

Scenario Definition

During the creation of use cases it is often useful to identify scenarios that are part of a use case, or to assign discovered use cases to another as a scenario. As noted above Scenarios are defined as a specified path through a use case, and may be the Basic, Alternate or Exceptional paths.

A typical way to capture scenarios is by using UML Activity Diagrams or other flow-based diagramming methods. UML Activity Diagrams are preferred since they fit well with other UML design documentation (permitting traceability from requirements to implementation), and the contain a richer set of relationships compared to simple flow-charts. These diagrams show the flow of activities, decision points, parallel activities, as well as additional information (such as required data element for an activity) – see Figure 3.

Scenarios should be used to highlight dependencies on external Actors (and often discover missing Actors), and to indicate the required data or information that is needed to perform the specified use case.

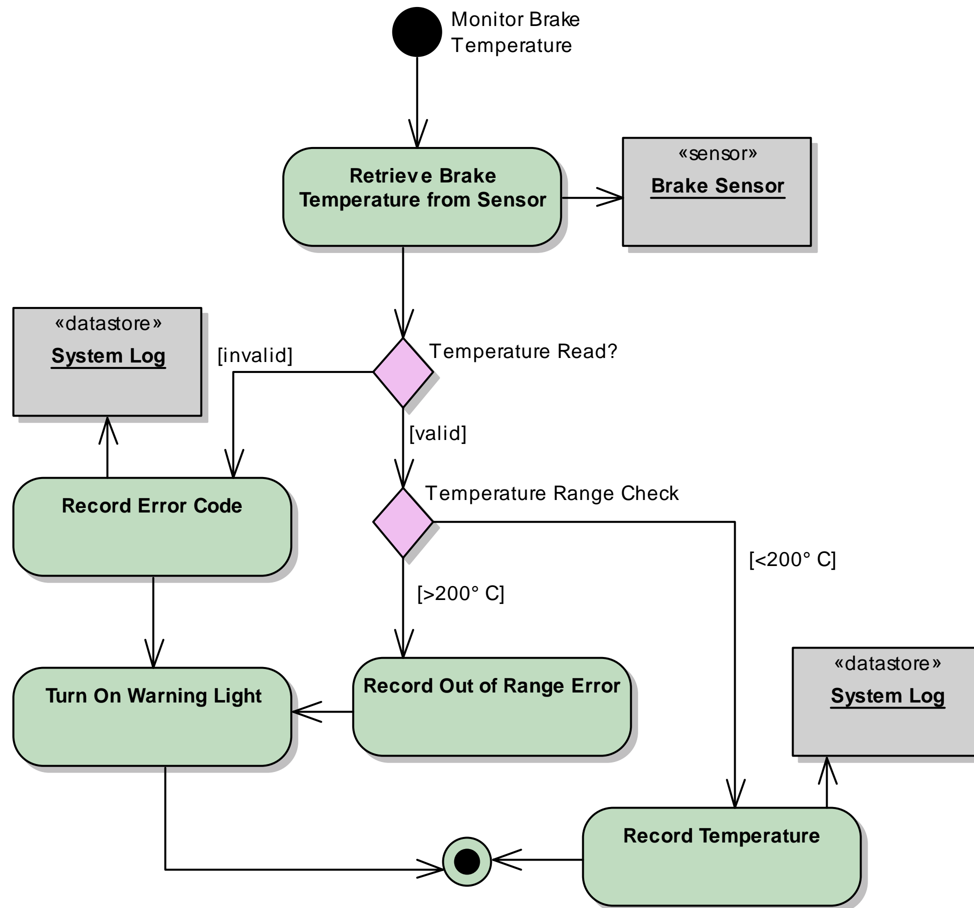


Figure 3. Use Case Activity Diagram - Monitor Brake Temperature

Refinement and Revision

Refinement is a critical part of use case development. A use case is rarely correct as first defined, and will often undergo multiple rounds of revision, collection, and fragmentation as part of the requirements definition process. As new scenarios are discovered they should be assigned to existing use cases or new use cases. As use case common functionality is discovered it will indicate the need to add new use cases or remove unnecessary ones. Much in the same way that an author needs to edit a story many times, a use case modeler must revisit the model many times. This interative approach to use case definition ultimately results in a model that is clear, complete, and more concise.

The refinement process need not be conducted while in Workshops, and may instead be done as an after-hours or off-site project. Usually, the use case modeler will review new information and incorporate it into the model. During this process, the modeler will

review the updated model with other members of the team and with the subject matter experts to verify that the changes are correct, and fit into the overall model structure.

Assignment

Describe and model your behavior for a typical daily activity, such as preparing for sleep, washing a car, preparing a meal, etc. Include all interactions with secondary system actors (e.g. stove, water supply, etc.). Detail use cases with Basic Flow, Alternate Flow, and Exceptional Flows. Deliverables: Use Case Model, Actors List, One detailed use case (all flows).

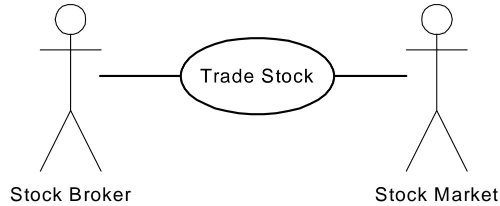
Review the example use case (Appendix). Note how data elements are described and modeled. How would you have presented this information?

References

1. Jacobson, I., et al., *Object-Oriented Software Engineering: A Use Case Driven Approach*. 1992, Harlow, Essex, England: Addison Wesley Longman.
2. Lieberman, B., *Putting Use Cases to Work.* The Rational Edge, 2002. **February**.

APPENDIX: Example Use Case - Stock Trader System

Use Case Definition: Trade Stock



Basic Flow:

1. The customer contacts the Stock Broker and requests a trade of a one or more stocks. The following information is captured by the Stock Broker:
    - Customer Name
    - Account Number
    - Stock Information (one set for each trade):
        - Stock Ticker Symbol
        - Trade Amount (number of stock certificates)
        - Conditions to Perform the Trade (i.e. Immediate, Buy Upper Price, Buy Lower Price, Sell Upper Price, Sell Lower Price)
2. The Stock Broker enters the information into the system and requests the trade to be executed
3. The system executes the trade on the appropriate market (i.e. NASDAQ, NYSE, etc.)
4. The system enters audit information into the Customer History record:
    - Customer ID
    - Account Number
    - Stock Information
    - Date (MM/DD/YYYY) and Time (HH:MM:SS)
    - Result of Trade (Confirm/Fail)
5. The system updates the Customer Portfolio information:
    - Customer ID
    - Account Number
    - Stock Information:
        - Stock Ticker Symbol
        - Current Holdings
            - Number of Certificates
            - Value of Stock
    - Last Date of Change
6. The system responds with confirmation of the trade
7. The Stock Broker relays the result information to the Customer

Alternate Flow:

At step 2 of the Basic flow the market may be closed, in which case the trade order is saved for execution as soon as the system detects the market is available:

1. The system retrieves the trade order from the database
2. The system processes the order request.
3. Use Case flow resumes at step 3 of the basic flow.

Exceptional Flow:
At step 3 of the Basic Flow, if the trade is not conducted due to technical reasons the system will log the error and report that the trade has failed:
1. The system retrieves the error code from the Stock Market return message (see Appendix for codes).
2. The system logs the failed transaction information to the Failed Transaction Log:
   Markent ID (e.g. NASDAQ, NYSE, AMEX, etc.)
   Customer ID
   Account Number
   Stock Information (see Basic Flow)
   Date (MM/DD/YYYY) and Time (HH:MM:SS)
   Reason for Failure
3. The system responds to the Stock Broker that the trade failed
4. The use case resumes at step 6 of the Basic Flow

Special Requirements:
- All trade transactions must occur within 2 seconds
- Trader System must be available 24/7/356
- All trade operations must be logged for audit purposes
- All Broker Actions must be tracked

PreConditions:
- A Client Profile exists and is valid
- Investment Instruments have been defined
- A Client Portfolio exists and is valid
- The Stock Broker has been authenticated by the system security

PostConditions
- A stock trade is executed

  or
- An error condition is detected

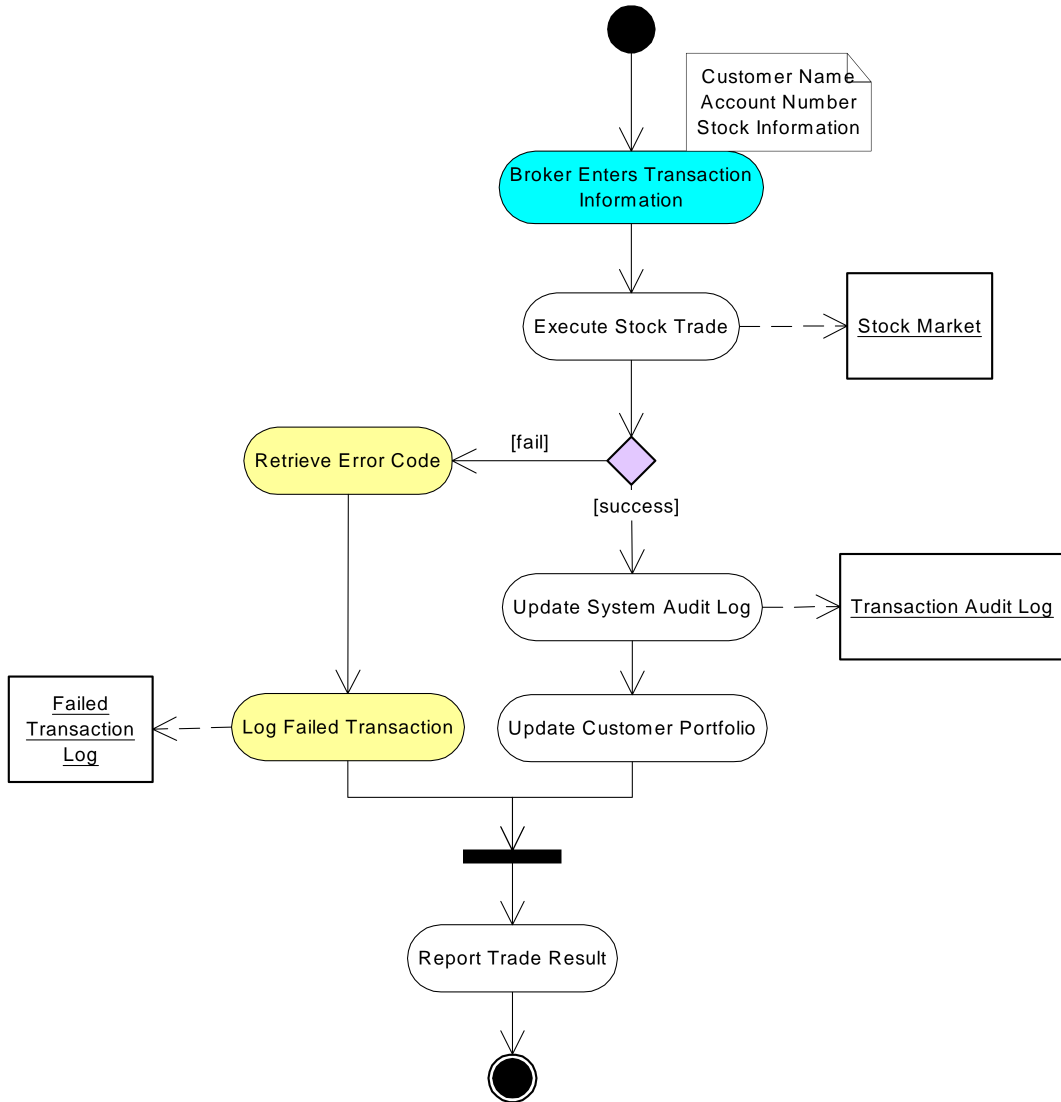


Figure 4. Trade Stock Activity Diagram