Using Use Cases for requirements capture

Pete McBreen
© ***1998 McBreen.Consulting***
petemcbreen@acm.org

Introduction

Developers have always used typical scenarios to try to understand what the requirements of a system are and how a system works. Unfortunately although developers have done this, it has rarely been documented in an effective manner. Use Cases are a technique for formalizing the capture of these scenarios.

Although Use Cases, as defined in the Objectory book (*Object Oriented Software Engineering, Jacobson, Christerson, Jonsson & Overgaard. Addison Wesley 1992*), are associated with Objects, the technique is actually independent of Object Orientation. Use Cases are an effective way of capturing both Business Processes and System Requirements, and the technique itself is very simple and easy to learn.

Making Requirements available for Review

The reason for formally capturing the scenarios is to make them available for review by both the users and the developers. There are 2 specific criteria that any functional requirements notations needs to meet:-

1) it should be easy for the sources and reviewers of the requirements to understand and

2) it should not involve any decisions about the form and content of the system.

Functional Requirements are external requirements that are to be used to evaluate the design and final implementation of the system.

What these requirements have to do is capture in an implementation-independent manner what the needs and expectations of the stakeholders are.

Use Cases make requirements available for review

Use Cases are starting to become widely used. Compared to other requirements capture techniques, Use Cases have been successful because:-

- Use Cases treat the system as a black box and
- Use Cases make it easy to see implementation decisions in requirements

This last point arises out of the first. A Use Case should not nominate any internal structure to the system that the requirements relate to. So if the Use Case states “Commit changes to Order Database” or “Display results on Web Page”, the internal structure is easily seen and can be flagged as a potential design constraint.

The reason why the requirements should not nominate any internal structure, is that specifying the internals puts extra constraints on the designers. Without these constraints designers have more freedom to create a system that correctly implements the externally observable behavior, and the possibility exists that they may come up with a breakthrough solution.

Industry acceptance of Use Cases

Use Cases were first formally described about six years ago, and since then have been adopted by most major object oriented methodologies. Use Cases have also been adopted by the business process reengineering community as a useful technique for describing how a business operates, for both the present and future modes of operation.

Use Cases themselves have recently received new prominence by virtue of their place and position within the unified modeling language (UML). The UML has been proposed to the OMG as the standard modeling language for object systems.

What are Use Cases?

A Use Case itself is an interaction that a User or other System has with the system that is being designed, in order to achieve a goal. The term Actor is used to describe the person or system that has the goal, this term is used to emphasize the fact that any person or system could have the goal.

The goal itself is phrased with an active verb first; examples being "*Customer: place order*", "*Clerk: reorder stock*".

As part of the Use Case it is necessary to document what goal success and goal failure means to the Actor and the System. Within the context of placing an order, goal success probably includes the goods being delivered to the Actor and the Company receiving the appropriate payment for said goods. Careful definition of Goal Success and Failure are essential for defining the scope of the system, since for a limited Order Entry system, Goal Success would merely mean that the order has been validated and delivery scheduled.

The Role of Scenarios

Different scenarios within a Use Case show how the goal succeeds or fails; a success scenario is one in which the goal is achieved, a failure scenario is one where the goal is not achieved.

The nice part about this is that because the goals summarize the intention of the various uses of the system, the users can see how they are supposed to use the system. Users can also spot when the system does not support all of their goals, without having to wait for the first prototype, or even worse having to wait for the system to be developed.

How "Big" should a Use Case be?

An interesting topic is that of trying to determine how to "size" a Use Case. One way to address this is to relate the size to the purpose and scope of the Use Case. With a really large scope, a Use Case is not so much addressing a single system but all of the systems used by a business. Such a ***Business Use Case***, would treat the entire company as a black box, and speak of the Actor's Goals with respect to the Company. The scenarios for these Business Use Cases would not be allowed to assume any internal structure to the company. A customer would place an order with the Company, not the Customer Service department.

For System Development however, the scope of the Use Case is restricted to a single system. These are the most common form of Use Cases, which we could call ***System Use Cases***, and they treat the system as a black box. These Use Cases cannot specify any internal structure and should be restricted to using only words from the problem domain.

Another scope for Use Cases is the design of sub-systems or components within a system. These ***Implementation Use Cases*** treat a component as a black box and the actors are the components that interface to it. For example it would be possible to use Implementation Use Cases to specify the requirements for an email component to be used by an application.

Given this classification, the conversation about the size of a Use Case is easier. The scope of the item to be designed sets the overall size. To assist the system designers, each Use Case should describe only a single thread of activities without major branches. Violation of this restriction can usually be seen by imprecise or vague Goal Success criteria, which makes it hard to use the Use Case as a source for test specifications.

As an example of a System Use Case, "*Query database for low stock*" is too small as can be seen by the mixing of the implementation detail with the requirements. By contrast however, as a System Use Case, "*Manage Warehouse*" is too large, since it cannot be accomplished as a single thread of activities without major branches, and from a system viewpoint it would be hard to

specify Goal Success. But it would make a good Business Use Case for a Parts Department. For a Parts Department, it would be possible to define Goal Success for "*Manage Warehouse*", (probably in terms of Inventory turns, parts availability, operating costs etc.)

The nice thing about these Business Use Cases is that they can be used to categorize the other Use Cases. Thus "*Manage Warehouse*" could be used to group all of the Use Cases involved with the actual management of the warehouse.

A Formal Definition of Use Cases

A Use Case a delivers a measurable result of value to a particular Actor.

As noted previously, Actors can be people or external systems that the system being designed has to interact with. The requirement for a Use Case to have a measurable result, is derived from the need for a single thread. As part of having a measurable result, the goal either succeeds or the goal fails. There is no room for middle ground.

Achieving the goal of the primary actor is defined as a success, all results that do not meet the goal of the primary actor are defined as goal failure. The different scenarios show all of the paths to success or failure.

Documenting Use Cases

The nice thing about Use Cases is that the scenarios can be documented with varying degrees of formality. Each scenario refers to a single path through the Use Case, for a particular are set of conditions.

Informal text descriptions can be used, but they are sometimes difficult to follow when there are multiple conditions and possible failures. The informal narrative style, however is very useful initially while trying to understand the requirements. Later on in the development of the Use Cases however, it is useful to use a more formal mechanism for documenting the Use Cases.

A **rough sketch** of the Customer: Place Order Use Case could be as follows:-

"*Identify the customer, check that requested goods are in stock and that their credit limit is not exceeded*"

A **structured narrative** format has proved to be very effective. What this format does is specify a sequence of Actor:Goal pairs for each of the scenarios. The way these are written down is to specify the simple success scenario first, as a sequence of Actor:Goal statements, all of which assume success of all previous goals. This sequence, as shown in the example, is the simplest scenario that leads to success.

1. Clerk: Identify Customer
2. System: Identify Item
3. System: Confirm Ship Quantity
4. System: Confirm Credit
5: Customer: Authorize Payment
6. System: Dispatch Order
Extensions
1a. Customer not found
1a1. Clerk: Add new Customer
3a. Partial Stock
3a1. Clerk: Negotiate quantity

The Use Cases consider the system that we are designing to be a single black box. No internal structure is recorded at all, and it can be considered to be a single Actor for the purposes of writing out the scenario. The Use Cases do not say anything about the internals of the system, only what goals the system will have and what goals it is responsible for handling.

Handling Goal Failure - Extensions

What needs to be identified next is that each of the steps above may fail. The conditions that may lead to failure need to be captured as extensions to the scenario. These extensions are dealt with by writing a partial scenario below the failure condition and following that partial scenario until it either rejoins the main track or fails.

This separation of the failure conditions makes the scenarios more readable. The primary success scenario is the simplest path through the Use Case, each step of the way, the actor's goals were successful. A separate listing of all of the failure conditions allows for better quality assurance. Reviewers can easily check whether all conditions have been specified, or whether some potential conditions have been omitted. Failure scenarios can either be recoverable or non-recoverable. Recoverable failure scenarios eventually succeed, non-recoverable failure scenarios fail directly.

Failures within Failures

One extra complexity that needs to be highlighted at this point is that within a failure scenario, other failures can occur. This means that the extensions section can have further Failures identified with a slightly longer prefix number: 1a1b. Customer is a bad credit risk. This would be recovered through 1a1b1 .

Why use a structured narrative format?

The value of the structured narrative format is that it is a refutable description. The value of having a refutable description is that it is precise enough to be argued over and disagreed with.

"*That is not the way it works*"

"*We do not check availability when taking orders*"

"*You missed a few steps*"

In contrast, the rough sketch provided by informal narrative text descriptions is hard to refute, but it is useful for early understanding of a problem domain.

Another way of describing the value of a refutable description is that when you document Use Cases, expect to get feedback from users and developers about the quality of the Use Cases. This is very valuable since it means that corrections can be made very early in the development process. The typical feedback from users highlights different sequencing, possible parallelism or missing steps. Typical feedback from developers is related to requests for clarification about what a particular failure condition means and how to detect it.

Graphical Notations for Use Cases

There is a graphical notation for depicting Use Cases. The UML uses a simple notation of a "Stick-Man" symbol for the actors and Ovals for the Use Cases as shown in Figure 1 below. These diagrams are great for when you want to see the "big picture" of how a set of Use Cases are related, and to get an overall picture of the context of a system.

Use Case diagrams do not show the different scenarios, what they are intended to show is the relationship between the actors and the Use Cases. So the Use Case diagram needs to be supplemented with the structured narrative text. The UML does have alternative diagrams to show the different scenarios, the normal diagrams that would be used are interaction diagrams and activity diagrams. The main drawback to the use of these diagrams is that they are not as compact as text, but they can be useful for giving the overall feel of a Use Case. For compact a reference on interaction diagrams and activity diagrams see "UML Distilled" by Martin Fowler (*Addison-Wesley 1997*).

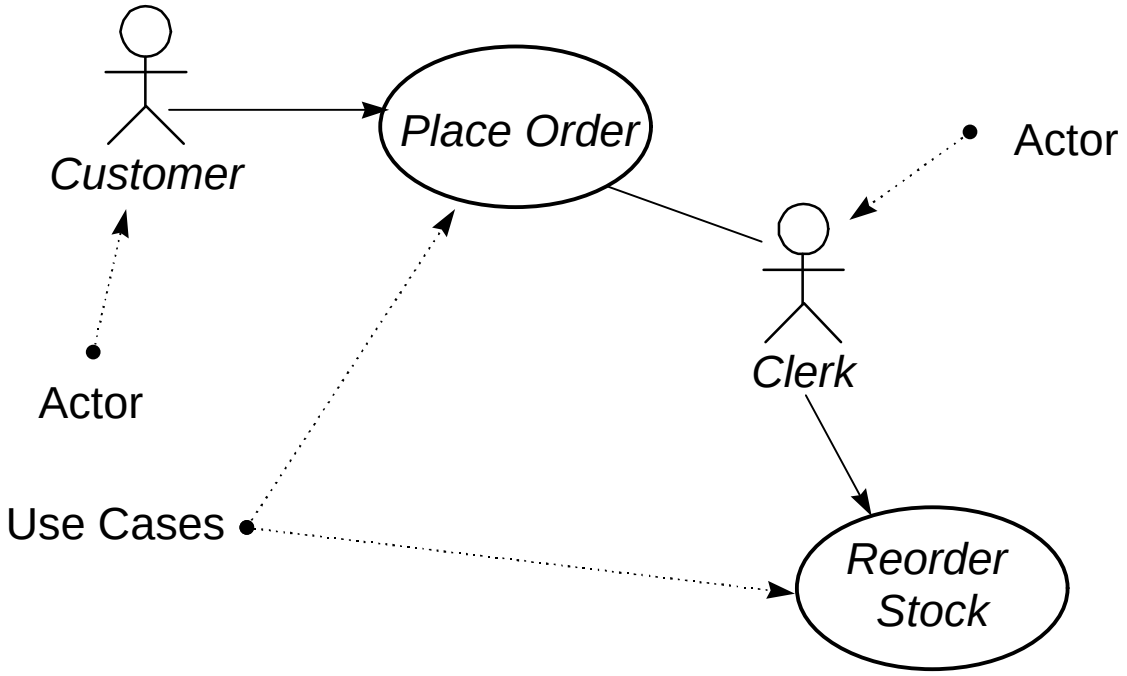

Figure 1.

Obtaining Requirements Reuse

The value of writing the Use Case descriptions in the Actor:Goal format is that it allows common functionality to be factored out into separate Use Cases. When this is done the commonality is said to be "used" by the main Use Case. For example the "*Identify Customer*" step of Place order could be "used" by several other Use Cases.

Another relationship between Use Cases is that of "extension", a Use Case is said to extend another Use Case, if it has complex failure recovery steps, normally with more than three or four steps to recovery. A possible extension to "*Place Order*" would be "*Issue Raincheck*" for the case when there is no stock available.

Complexities and hazards in applying Use Cases

No connection between Primary Actor and Use Case

In some cases, there is not a clear connection between the actors who get value from the Use Case, and the actors actively participating in the Use Case. For example the Chief Accountant could be the actor for "*Issue Invoices*", but it is unlikely that they would actually initiate the invoice run. This is not a problem, the Use Case is still valid, it just means that the connection between the actors getting the value and the initiation of the Use Case happens outside of the scope of the system being designed. The primary actor is still useful because the person playing that role is the person you need to talk to when documenting the Use Case.

Scenario steps do not need to be sequential

In cases when the sequence of steps in the scenario does not matter, there are several mechanisms to highlight the potential parallelism. Activity diagrams are the preferred mechanism within the UML, but informally you can notice potentials for parallelism by looking at the Use Case scenario and seeing those steps that are adjacent within the Use Case, and have the same actor responsible

for the step. Thus in the example we had earlier, there is a potential to confirm the ship quantity and confirm credit in parallel. Sometimes it is useful to note this potential parallelism in the Use Case documentation.

Sizing the Use Cases

A definite hazard when starting doing Use Cases, is to either have to many steps or too few steps. If there are more that 15 steps in the Use Case, it is likely that some implementation detail has been included. If there of very few steps, check to see if the goal can be achieved as a single thread of activity without many branches.

Few (if any) Human Actors

If there are few human actors, and most of the Use Cases come from other systems, then the normal mechanism for identifying Use Cases has to be modified. Rather than interviewing the actors, (this is somewhat hard to do with a machine), instead look for the external events that the system must react to or recognize.

Requirements Capture and System Complexity

In review, the scenarios capture system complexity while the Actor:Goal pairs enable even large systems to be documented in a relatively condensed format. The value of this Use Case format is that instead of having a large functional specification that is rarely read, with Use Cases, users and developers can identify the actors and then confirm that the listed goals match (or do not match) the job responsibilities of that actor.

Only then, for the Use Cases that the user or developer is interested in, is it necessary to dive into the details of the scenarios.

But Systems have more than just Functional requirements

Use Cases however, do not capture all of the external requirements of a system. Use Cases merely capture the functional requirements of how the system will be used. There are many other aspects to the requirements that must be captured and addressed. Some of these non-functional requirements are however, use related, so they can be attached to an individual Use Case. Examples of this include requirements such as throughput and performance. Other requirements however are not use related and need to be captured separately. Examples of these would be

- System scope
- the Goals that the Users have for the system
- user interface prototypes
- general rules
- constraints
- algorithms

Run Time vs. Build Time Requirements

An important factor to remember when capturing requirements is that the constituency for the system is much larger than just the User community. There are many different Stakeholders in the system, and Use Cases only capture the needs of some of these Stakeholders. In essence, the Use Cases only capture the run time requirements for the system and ignore a major Stakeholder, the system development organization. The Development Organization is has a strong interest in specifying the build time requirements for the system.

Run time requirements include: System scope, Goal and expectations of the product in the user organization, Use Cases, Other non-functional requirements.

The build time requirements include: Ease of development, Robustness in the face of change, Reuse of existing components.

The build time requirements can partially be handled by Use Cases. But the are many other aspects that the development organization needs to address;

- **Project scope and goals:** what this project must deliver (as distinct from the system scope which typically will be delivered over several projects)
- **Instances of growth and change:** these can be captured as "Change cases" in the normal Use Case format
- **Development sponsor constraints:** these include standards, practices, tools, quality measures and quality assurance principles and practices.

Applicability of Use Cases

Use cases are primarily for systems that need to respond to external events. They can be used in other environments provided that there is a clear actor who has a clear understandable goal.

Use Cases cannot be used when the outcome is undefined or unclear.

This means, that if goal success and goal failure cannot be defined, then Use Cases should not be used to capture the requirements.

Having said that however, most modern object methodologies are now using Use Cases. This is because Use Cases have proved to be a very effective mechanism for capturing requirements.

Summary

Use Cases capture requirements in a readable, refutable format. The Use Cases are a refutable description of the externally required functionally of the system.

Refutable means that when you document Use Cases, expect to get feedback from users and developers about the quality of the Use Cases.

Use Cases do not need to be precisely defined right from the start. The typical development sequence is as follows

1. Identify actors.
2. Identify the goals of the actors.
3. Identify what success and failure means for each of the Actor:Goal pairs.
4. Identify the primary success scenario for each of these Use Cases.
5. During elaboration, identify failure conditions and the Recoverable/non-Recoverable scenarios.

It is only necessary to get to step 4 before deciding which releases of a product a particular Use Case will be developed in.

In summary, Use Cases are an effective requirements capture technique that makes requirements available for review, avoiding any implementation bias in the requirements.

Acknowledgments

This article was heavily influenced by “Structuring Use Cases with Goals” by Alistair Cockburn, see http://members.aol.com/acockburn for the original reference and a suitable template for documenting Use Cases.