JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2005**

Vol. 4, No. 6, Special Issue: Use Case Modeling at UML-2004, Aug 2005

Formalism, technique and rigour in Use Case Modeling

Bruce Anderson, Application Innovation, Business Consulting Services, IBM

Abstract

Use case Modeling is widely used as a technique for requirements gathering but does not always lead to clear agreement between users and developers, or to effective system development. This is often because the model does not have a clear role in a clear process, with a corresponding lack of agreed standards and techniques. Taking a considered approach and tailoring the available guidance to the situation at hand can produce more appropriate use cases that are more useful in the overall process. This paper outlines a sound approach in a context of ideas and technique, and discusses several common issues in use case modeling, with suggested resolutions.

1 INTRODUCTION

Use Case Modeling has become a vital technique in system development, yet there are many remaining questions of many kinds in developing powerful such models and using them in the construction of appropriate systems.

This paper begins with a distillation of some of my experience in requirements engineering and Use Case Modeling over a number of projects of varied sizes and different business domains. It then goes on to list some of the issues (both those I see and those that have been presented by others), together with some suggested resolutions and ways forward.

My interest is in capturing and formulating good practice, and stimulating discussion, in order to assist practitioners. This does not preclude theory; quite the contrary "There is nothing so useful as a good theory." (Lewin 1951). The intent is produce a clear and consistent approach.

Note that, although we talk of requirements, a Use Case Model is in fact a specification at some level of detail and accuracy.

Although we talk of analysis, in fact synthesis is the key; those involved are co-creating a specification that satisfies the client.

The approach and guidance here is meant to be appropriate for any kind of project lifecycle, but most of my detailed experience is of rather conservative projects, which are

Cite this article as follows: Bruce Anderson: "Formalism, technique and rigour in Use Case Modeling", in *Journal of Object Technology*, Vol. 4, No. 6, Special Issue: Use Case Modeling at UML-2004, Aug 2005 , pp. 15-28.http://www.jot.fm/issues/issue_2005_08/article2

iterative and incremental, but somehow subject to a pull (often from the client) to somewhat waterfall and artefact-heavy processes.

Any practitioner, and any organisation, must improve their approach continuously, so all the ideas and suggestions here remain open for deeper understanding and refinement.

2 USE CASES, USE CASE MODELS, AND THEIR ARTEFACT CONTEXT

A use case is a narrative of the use of the system by a user, with alternating system and user actions, and a focus on what the user does and sees at some abstract level. It contains alternatives generated by different user behaviour, data values, or business rule results. The use case describes an attempt at achieving a business goal for the user. See Appendix D for a detailed template, which is in the style of [Cockburn01].

In general in a project we need to agree a functional specification with the client. In certain cases, a Use Case Model will be very suitable for this purpose. The client can understand the stories in the use cases (to see that the system will do what they want, to achieve their business goals), and architects and developers get enough detail for their understanding (to see that they can build it).

The Use Case Model is just one artefact in the functional requirements model of such a project; we also need a Logical Data Model (which the use cases refer to), a Business Process Model (to show how the use cases are used in the larger context - see the example in Appendix A), some Business Rules (to explain calculations and constraints), and some Business Scenarios (to show the business processes and use cases in action with illustrative values, thus helping people appreciate the whole picture).

We also need Non-Functional Requirements, but these, and their links to functional requirements and architecture, are not dealt with in this paper.

A use case has a "one person, one place at one time" scope, namely a task (node) in the business process model. In general we model task details with use cases if there is any possibility that they will be automated. A use case can always be executed by people if needed. The use case template in Appendix D provides good guidance in documenting such tasks and their steps.

3 A USE CASE MODEL CONTAINS FOUR PARTS

Preamble: introductory text giving the general approach and assumptions of the model

> *example content: "the cancel operation may be executed at any time; the use case will then terminate with no effect on the business state"*

Actor list: list of the roles that users and other systems take in using the system, with some details

example content: general assistant (does all shop-floor and warehouse tasks), manager (has extra actions for authorising and verifying), warehouse management system (updates order status)

Use case list: list of the use cases in the model, organised and classified in some way

example content: Pricing (reduce item price, make item label), Layout (record product mapping)

Use cases: the actual use cases. Each use case is expressed in the template.

4 PROJECT PROCESS, ORGANISATION, AND CONTEXT FOR USE CASES

A project has a network of artefacts as work-in-progress, related by dependency. The artefacts are each structured by templates. The project process breaks the project work into phases, and describes how each artefact is built up over time (in particular, by phase), giving level of detail, and accuracy. For the Use Case Model, this breakdown must also be made at the level of individual use cases.

for example, in the Rational Unified Process's inception phase [Kruchten98], which primarily deals with business risk, we list the actors and use cases, and generally outline some of the use cases, along with an outline data model and scoping of the business rules. See Appendix B on phase criteria from the IBM Global Services Method for a more detailed example

The project process must be written down clearly, tried, and improved continuously.

for example: does the code formalism for business rules work? can developers work accurately from the use cases? how are system test cases derived?

Designing this process, and getting it adopted, is a major part of any successful project. Though of course based at least on past projects, and probably on some systematic method, it requires experience, and iteration. It is the precursor to any planning.

One key principle is to make sure the use cases are being structured to fit what is needed for development, so that there is a clear route to code. This will require the architecture work to have established the technical components of the solution. It may well be that decisions on modularity and UI will allow a more form-filling approach to the functional spec.

for example, once the system metaphor and UI style is decided, with guidelines on navigation, we must make sure the use cases have matching granularity and have expression which fits that navigation

5 HOW RIGOROUS / FORMAL SHOULD THE USE CASE MODEL BE?

It is absolutely essential for the level of rigour to be understood, agreed, and maintained.

> *example: in Extreme Programming [Beck00], the Use Case Model is a temporary informal note used in the construction of the code, which is the rigorous spec*

If the Use Case Model is to be used as a spec (i.e. used by developers without a great deal of further contact with analysts or users) then it should be refined until it is essentially executable. This can be a "sloppy execution" where details will be filled in during coding, but often the model should be pretty detailed, for example in terms of the data that is used and the business rules. This refinement will take place in the various phases of the project, according to the agreed completion criteria for the artefacts worked on in each phase.

A primary force on these decisions about rigour is how the system will be maintained into the future. In practice at most one model can be maintained along with the code, whatever the standards or methods in use may say. These decisions will be derived from the non-functional requirements about flexibility and maintainability.

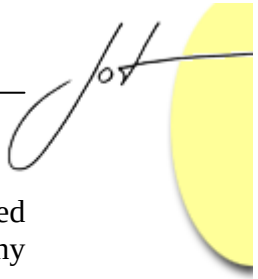
> *for example: we need to change the system's behaviour when a payment is made to a non-existent account. Where do we look first? There must be something we can read and understand, so that we can debug and maintain the system. This could be some very structured code with comments, or a separate Business Rules Catalogue with traceability to code*

6 A RIGOROUS-ENOUGH APPROACH TO THE USE CASE MODEL AND ITS NEIGHBOURS

Use cases affect business state (use case as black box)

A use case can be seen as an executable item in the world of the specification model, where the execution changes the business state. By describing these changes with preconditions and postconditions we can work with use cases at a level of abstraction that does not look at the steps.

- If the preconditions are true
  - if the use case is executed successfully, then the business state is affected so that the postconditions are true
  - if the use case fails, then the postconditions are not guaranteed, and the failure information may be used in alternative paths
- If the preconditions are false
  - there is no guarantee of anything. In such cases the use case should not be executed, and the system should be designed so that this doesn't happen



In our template we call the postconditions "outcomes", since writers and readers preferred that term. If the outcome is a failure then the reason is described, with a note of any corresponding action in the use case (and thus change in business state).

Note that the information about a failure reason is in what we might call the process or transaction state, to be dealt with by the context of use (a step in another use case or the business process), which may then set business state.

Use cases can be composed (sequential)

We use one use case (or more) to ensure the preconditions for others, so that we may see that the way the business processes use the use cases is correct. There is of course a good deal of design work to be done here, to ensure an efficient approach.

> *example: do Transactions validate themselves as a first step, or do we make sure that a Transaction is validated separately before it is executed? We must choose between [Do Payment] and [Validate Payment Request, Execute Payment Request].*

Use cases can be composed (hierarchical)

A use case step can be carried out by another use case. By analogy with code (as a description that can be run in a context of values), we refer to this informally as "calling" (as of a subroutine), just as we talk of "executing" a business process or use case. UML2 (and this paper) refers to this "calling" relationship as "includes" which is unfortunate since in IT this generally means textual inclusion (as of a macro).

It's important to know how a use case can fail, as each of these failures will lead to an alternative flow in the using use case.

> *example: "Validate delivery address" can succeed, fail with non-existent address, and fail with an address out of the delivery area; a using use case may want to take different actions in the two failure cases; this is why a table of such failure types is part of our use case template*

We might say - a use case step can be specified by an included use case.

The Logical Data Model is key to requirements work

The business state must be represented in terms of a data model that encompasses the whole system. Often it is useful to associate a glossary with this model, giving the various names used for items in the different parts of the business. Without such a model is it hard to see that the use cases are consistent with each other, and of course hard to move to development.

Business Rules must be executable and tied to a use case step

If a rule is recorded as a constraint (a requirement), then it must be re-expressed as an executable rule (a specification), and connected to the use case steps that maintain it at appropriate times.

example: suppose A+B must always be 7; then whenever A or B is changed, the rule must be invoked - this could be done by having a "Change A" use case (and a "Change B" one), or using a more generic "Change value" use case, or by just making sure that any step affecting A or B invoked the rule. If constraint-satisfaction was a major part of the solution, then it would be possible to have an executable constraint-maintaining framework, described in the Preamble (part of the abstract architecture) and then having an "Add constraint" use case.

A Business Event List is a useful artefact

It may be used as in index, and as a checklist, but note that it provides entry points to the Business Process Model, not the Use Case Model; the connection to use cases is via the tasks they support.

7 ISSUES IN USE CASE MODELING

In this section we deal with some of the problems that commonly arise, and the questions that are commonly asked, in creating and working with a Use Case Model.

Confusing the Use Case Model with a use case diagram

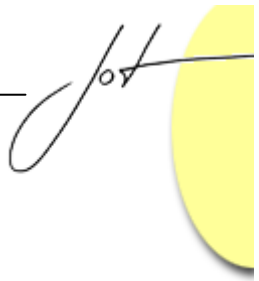
- A diagram of actors and use cases, with various use case relationships, can be useful, but is essentially just a visual index into the Use Case Model. A structured list may be better than a diagram.
- Such diagrams cannot show dynamic behaviour
- The use of <includes> (our "calls") is safe, but <extends> must be avoided unless its meaning is clearly documented. This requires understanding how and when to use it, template extensions, and being clear on the implications for development. There is no single clearly-accepted standard in this area.

Omitting sequence information and dynamic behaviour

- It is very helpful to see use cases in a larger context, usually in a diagram indicating how the business process uses use cases, to show the sequences in which they may be used
- It may be useful to use business scenarios, perhaps illustrated with sequence diagrams (with use cases and the business process represented as objects, and use of included use cases represented as messages) to show how use cases interact to achieve larger goals

Use of use cases when another formalism would be better

- There are times when a state machine or activity diagram is more appropriate. In particular, complex flows (such as an automated process may use) are better expressed as activity diagrams.



Inappropriate (limiting) control information in use cases

- A use case must not say which use case is next, or where it is called from. This leads to poor modularity - the information belongs in the Business Process Model or the calling use case. People with coding experience will understand this easily.

Intrusion of User Interface detail

- User interface detail must be kept out of the functional spec, but needs to be linked to it. Sometimes overall User Interface Guidelines plus simple annotation of the use case is enough as screen design documentation. This is an area that needs exploration, with clear possibilities for generation of the user interface from such a representation.

Object orientation and analysis modeling

- Use Case Modeling is not an object-oriented technique, and poor use case work should not be bolstered up by some analysis object model with classes and responsibilities created in some fanciful way that may or may not be useful in design and development. It is the Use Case Model that the client will see and agree, and that developers should be able to work from.

Parameterised / generic use cases

- A use case may be activated in different contexts, for example a worklist tool may start up with differing default search criteria. We need a clear way to say what in the calling context is available in the called context, and one technique is to use an explicit parameter. This is not business state, but process/transaction state, whereas the worklist itself is part of the state.

example: Maintain worklist (with Search Criteria)

- A related question is how to deal with what is essentially metadata. For example, the use case "Provide quotation for insurance product" will require different behaviour at the detailed level for each product, but can have a generic structure of steps which ask for information, calculate policy cost and so on. In practice we don't want a new use case for each product, so this is a good approach, but requires appropriate extension of the notation.

example: see Appendix C.

Local/global and the architectural influence

- The architecture will influence the shape of the Use Case Model, but this link is entirely informal. Someone with technical skill will contribute to the Preamble and to the modification of the use case template and the guidelines. The vision

and some of the understanding that support this must be clear to the practitioners who lead the use case work, and must be communicated to the whole team. If this is not done, the use cases may turn out to be incompatible with the architecture, requiring rework of either the use cases, or of the architecture.

example: the way the system deals with concurrency will affect the how users manage their tasks and how the Use Case Model is structured. Do blocked tasks just have different status in a task list ("waiting for your attention"), or interrupt current processing?

System actions and business state

- It is important not to prejudice the content of the solution (and the ability to change it) by putting implementation detail in the use case. The correct level of detail is to restrict the detail to changes in business state, for example using "... the system updates the Customer Details" in a step. But suppose we have "... the updated Customer Details are posted to overnight batch update", implying they aren't available until next day. If working with the batch system is a given, then this should be described in the Preamble, and the effects taken into account. This does require the Logical Data Model to include the batch, and the business state to include that of the batch.

Layering the Use Case Model

- It can be useful to layer a business process model, from a list of high-level ones with each broken down into lower level ones, which are again broken down. For a use case model the "one person, one place, one time" scope means that three levels is the most that is useful, and only when the lowest is simple subfunctions such as recognising a product on a shelf.
- My own rule of thumb is that if you have more than 80 use cases for a system (which might be a subsystem) then you need to work with more generic use cases, for example moving from "Change Customer Address" and similar use cases to "Maintain Customer Information". If this doesn't work then the system needs some restructuring into subsystems, or perhaps even rethinking altogether.

Sequence, detail and accuracy over time

- Practitioners often find it hard to think of use cases as other than either "finished" or "not finished", and then too often thinking that iteration means "it's OK to finish it later". A project must have clear phases, with clear criteria for completion. It can be useful to name intermediate states, so that for example having the name, goal and pre/post conditions is called "outlined". Some criteria for phase completion are given in Appendix B.

Needing a vision

- A team can make use cases only for a system that they can imagine already. At the stage where anyone (as end-user or developer) is wondering what kind of system will fit the bill, some envisioning work is needed. This is a quite different kind of activity from gathering requirements.
- Envisioning is definitely part of requirements engineering, or at least needs to be tightly integrated. After all, the vision will be refined during the Use Case Modeling, as we synthesise the specifications.

8 CONCLUSION

Despite the ubiquity of use case modeling as a concept, there are very many ways to approach it as a practical activity. One key to success on a project is to make concrete and public decisions on many aspects of structure, technique and method. We have indicated here some of the questions that must be surfaced, and a corresponding framework of ideas and practical answers that have worked well. The answers on other projects may differ, but asking and answering the questions is essential.

9 ACKNOWLEDGEMENTS

Thanks to Alan Miller, who got me thinking, Alistair Cockburn who gave me a good start, colleagues Paul Fertig, Terri Lydiard, Lynn Shrewsbury and Christoph Steindl in IBM, and of course all the students on our courses, and clients of our methods.

Thanks also to the anonymous reviewers of the paper who pointed out several inconsistencies and suggested many improvements, and to Benedict Heal who motivated many changes for the better.

REFERENCES

[Beck00] Beck, K: *Extreme Programming Explained: Embrace Change*. Boston: Addison-Wesley (good introduction to XP)

[Bittner & Spence03] Bittner, K and I Spence: *Use Case Modeling*. Boston: Addison-Wesley (very clear that modeling is synthesis, and on the need for a vision)

[Cockburn01] Cockburn, A: *Writing effective use cases*. Boston: Addison-Wesley (much of my thinking is derived from Alistair's work)

[D'Souza&Wills99] D'Souza, D and A Wills:. *Objects, components and frameworks with UML*. Boston: Addison-Wesley (very clear on business state, pre/post-conditions)

[Kruchten88] Kruchten, P: *The Rational Unified Process: an introduction.* Boston: Addison-Wesley (good introduction to RUP)

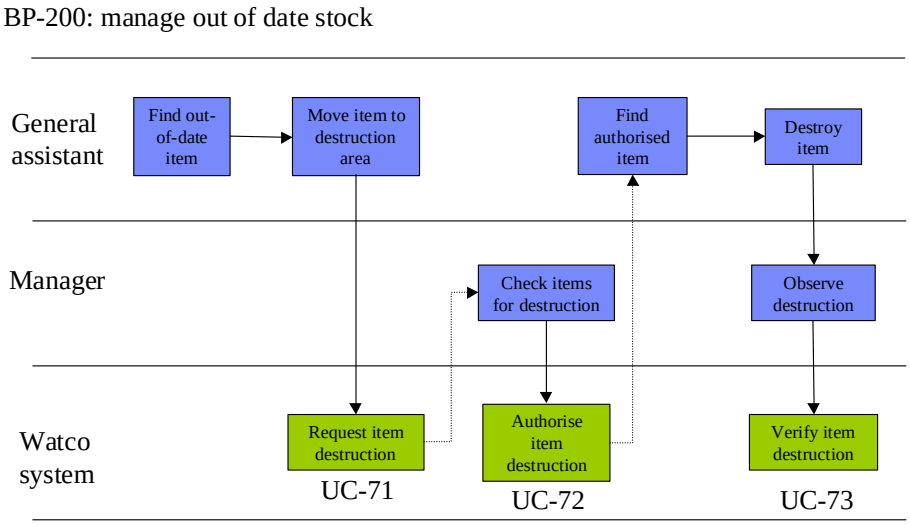
[Lewin51] Lewin, K: *Field theory in social science*. New York: Harper Row

About the author

Bruce Anderson is a managing consultant in IBM's Application Innovation practice. He is interested in finding the abstractions and shared maps and models that let a whole team understand what's needed on a project, from broad-brush business concepts to detailed templates for work products. Sometimes he work as a method consultant or requirements engineer; currently he is an application architect designing the enterprise service model for a large UK government department. E-Mail: Bruce_Anderson@uk.ibm.com.

10 APPENDIX A - BUSINESS PROCESS MODEL TIED TO USE CASES

The swimlanes for General assistant and Manager show business tasks, supported by use cases in the Watco system swimlane. In this piece of process each task is supported by one use case at most, whereas sometimes several are needed.



11 APPENDIX B - EXAMPLE CRITERIA FOR USE CASE MODEL COMPLETION BY PHASE

The technique of use case modeling may appear purely waterfall, but it isn't, or rather it needn't be. Use cases are typically modelled in an incremental manner, with a business process-worth of use cases being modelled at one time. It is necessary to prioritise business processes for use case modeling, and it may be decided not to model certain use cases, or perhaps only to model them down to a particular level of detail. The missing detail may be added in later phases, or perhaps not at all. Of course, if detail is omitted it may be appropriate to record a project risk.

Iteration in the Custom Application Development (CAD) engagement models

The CAD models of the IBM Global Services Method (GSM) are inherently iterative. Work-products are started in Solution Outline, progressed in Macro Design, and completed in Micro Design in each release cycle. Here are suggested completion criteria for use cases at the end of these phases.

Actual completion criteria will depend on the nature of the specific engagement, but should be clearly spelled out early in the project.

The word "key" below means "chosen in order to minimise risk". Typically that will mean "architecturally significant", but it might mean "most persuasive for the client", "scope uncertain" or "least understood for detailed requirements gathering".

Solution Outline

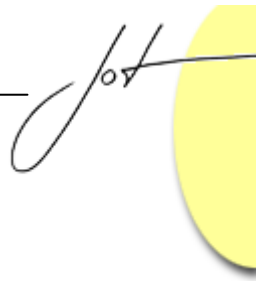
- Goal of phase
  - To provide client and project management with the necessary cost, schedule and risk information to make an informed investment decision regarding a potential new system.
- Criteria
  - All actors identified and documented with one-liners
  - All candidate use cases identified and documented with Goal in Context, Preconditions and Success Conditions, classified by business process or functional area
  - Main scenarios defined for key use cases
  - Business failure conditions of key use cases
  - Full template completed for the use cases whose design and implementation is considered to carry significant technical or financial risk
  - Preamble drafted, with complete outline and significant content on key topics

Macro Design

- Goal of phase
  - To develop a robust architectural framework upon which to build agile releases.
- Criteria
  - Main scenarios defined for all use cases
  - Alternative scenarios of key use cases
  - Business and IT failure conditions of all use cases
  - Full template completed for at least one end-to-end sequence of use cases within each business process
  - Full template completed for the use cases whose design and implementation is likely to raise significant architectural issues
  - Preamble complete

Micro Design

- Goal of phase
  - To prepare for the build cycle of a specific release of the system by driving the architecture and design to a release-specific and implementation platform view.
- Criteria
  - Full template completed for all use cases to be implemented within the release



- Preamble refined if necessary

12 APPENDIX C - EXAMPLE OF GENERIC USE CASE

- Main success scenario

...

1. System shows loan types
2. User selects a loan type
3. System requests loan application information
4. User enters loan application information
5. System shows warning notices
6. User acknowledges warning notices

...

- Related information/Generics

loan type: personal loan, flexiloan
loan application information:
for personal loan: amount, term, ...
for flexiloan: ...

APPENDIX D - USE CASE TEMPLATE

Use Case <Number> : <Name of use case>

Each use case in the project has a unique **number**. The **name** is an active verb phrase, the goal of the primary actor, and something that the system can deliver to the primary actor.
e.g. Use Case 22: Purchase Item

Version	Issue date	Author	Changes

Scope & level
The **scope** describes the system the use case is for, the boundary. The **level** is primary i.e. a significant business goal for the primary actor, or subfunction, i.e. a smaller goal.
e.g. Investment Planner - primary task, or Shelf Edge - subfunction

Goal in context
Explains the business **goal** and the **context** in which it is used, the state of the world as the use case will find it.
e.g. The user wishes to delete data as specified in a file in XML format

Preconditions
The state of the system when the use case starts. The system should guarantee that this use case won't start unless these conditions are met. Therefore, you will not check these conditions again during the use case.
e.g. The agent is authorised

Successful outcome
The state of the system on successful completion of the use case, i.e. achievement of the goal.
e.g. The Purchaser's account is debited

Failure outcomes
The state of the system if the use case fails. These possibilities are captured in a table

Failure	Outcome

where an **outcome** is associated with each "the use case ends in failure" step in a **scenario**.
e.g. Failure: customer ineligible; Outcome: registration attempt noted for MI purposes;

Primary actor
Who has the goal? Must be outside the system named in Scope.
e.g. Financial Planning Manager

Supporting actors
Other actors or systems involved in the use case.
e.g. A Supervisor or Credit Card System

Main scenario
A likely, common or interesting success path.
This is in the format
<step number> <action>
where **action** is of the form <subject> <active verb> <object>. Make sure the actions moves forward after this step. After each step, another subject "has the ball", so that typically we bounce user-system-user.
e.g. 3. The system displays a list of matching customers and requests a selection

Alternatives
The alternatives steps for the main success criteria are documented here in the following format <alternating step number> <branch letter><condition>:, followed by <alternating step number><branch letter><step number><action>, where **alternating step number** is the step at which **condition** occurs and we follow an alternative sequence of numbered steps. One possible action is "the use case continues at step <step number>".
e.g. 3a. There are no matching customers:
3a1. System reports that no match was found

Variations
Where there are several similar alternatives, rather than document each one, these variations can be documented here. List the steps involved and the nature of the variation.
e.g. The option " Favourites" is the same except that only a pre-selected five descriptions appear in the generated list

Related information
Various information can be kept here, and the sub-headings used should be decided at the beginning of the project. Typical categories are priority, data descriptions, business rules, system actions
e.g. Data definitions: existing holdings = Company, Holding Name, Price, Quantity

Issues
List of outstanding issues associated with the use case. Project issues should be documented separately in a project issues list.
e.g. 2. Number of alternatives needs checking with Compliance