Formalism, technique and rigour in use case modelling

Bruce Anderson

Application Innovation, Business Consulting Services, IBM
Bruce_Anderson@uk.ibm.com

Preamble

In this paper I distil some of my experience in requirements engineering and use case modelling, then 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.

This is not an academic paper - it contains few references, and there is no quest for novelty. 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).

Note that although we talk of requirements, a use case model is in fact a specification at some level of detail and accuracy.

And although we talk of analysis, in fact synthesis is the key - co-creating a specification that satisfies the client's requirements.

My detailed experience is generally of rather conservative projects - iterative and incremental, but somehow subject to a pull (often from the client) to somewhat waterfall and artefact-heavy processes. This is permitted, but not forced, by my proposals in the paper.

I'd like to be clearer on the distinction between technique and formalism, though I'm not sure there is a clean border there.

In this paper I present my approach rather definitely, but of course I will discuss and improve any of the points I give.

Use cases, use case models, and their artefact context

For me 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. (A template is included in an appendix).

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 it will do what they want), and architects/developers get enough detail for understanding (to see that they can build it).

The use case model is just one artefact in the functional requirements 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 example in appendix), some business rules (to explain calculations and constraints), and some illustrative scenarios (to show the business processes in action and help 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 - a task (node) in the business process model. Don't model task details except with use cases if there is any possibility they will be automated. A use case can always be executed by people if needed.

A use case model contains four parts

- Preamble: introductory text giving the general approach and assumptions of the model
– example: "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 take in using the system, with some details
– example: general assistant (does all shop-floor and warehouse tasks), manager (has extra actions for authorising and verifying)
- Use case list: list of the use cases in the model, organised and classified in some way
– example: Pricing (reduce item price, make item label), Layout (record product mapping)
- Use cases: the actual use cases

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 describes how each artefact is built up over time (in particular, by phase), giving level of detail, and accuracy. In the use case model, this is also by individual use case. We also need to say whether each artefact is maintained or a throw-away artefact.
– for example, in RUP's inception phase, 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 the appendix on phase criteria from the IBM 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?

Producing this process, and getting it adopted, is a major part of any successful project. 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, based on the architecture. It may well be that decisions on modularity and UI will allow a more "form-filling" approach to the functional spec.
- for example, in a system using a workflow product, there may well be a tool that captures the business processes and turns them into executable code

How rigorous / formal should the UCM be?

The first, and absolute, essential, is that the level of rigour must be understood, agreed, and maintained.
- example: in XP, the use case model is a temporary informal note used in the construction of the code, which is the rigorous spec

My own view is that the use case model 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 UCM should be a pretty detailed specification.

A primary force on these decisions is how the system will be maintained into the future. Only one model (at most) can be maintained along with the code. These decisions will be derived from the non-functional requirements about flexibility and maintainability.
- for example: we need to change the 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 traceabililty to code

My "rigorous" approach to the UCM and its neighbours

Use cases affect business state (UC as black box)

A use case is "executable" in the "world of the specification model", so that the preconditions and the postconditions of a use case are very important. Basically, a use case affects the business state:
- 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; of course 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". In the failure case the reason is described, with a note of any corresponding action in the use case.

I am aware that in the case of use case failure I don't usually say very specifically what happens to the business state - I think I expect that it isn't changed (unless noted), and that the information about the failure is in the "process" or "transaction" state, to be dealt with my some alternative step in the calling use case (or business process framework), which may then set business state. This needs clarifying.

Use cases can be composed (sequential)

We use one use case (or more) to "set up" 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 does the calling use cases (or business process) do that separately? We must choose between [Do Payment] and [Validate Payment Request, Execute Payment Request].

Use cases can be composed (hierarchical)

It's important to know how a use case can fail, as each of these failures will be an alternative flow in the "calling" use case.
- example: "Validate delivery address" can succeed, fail with non-existent address, and fail with an address out of the delivery area; a calling use case may want to take different actions in the two failure cases; a table of failure types is part of our use case template

This can be seen as - a use case step can be carried out by a use case.

Logical Data Model is key

The business state must be represented in terms of a data model.

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

If rules are collected ("requirements") as constraints, then they must be re-expressed (specifications) as the executable rules, connected to use case steps, that maintain that constraint 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, or by having some constraint-maintaining framework, described in the Preamble (part of an "abstract architecture") and then having an "Add constraint" use case

Business Event List

This is a useful artefact, but note that it provides entry points to the Business Process Model, not the Use Case Model; the tasks in the BPM may well be supported by the system, via Use Cases.

Issues in use case modelling

Overemphasis on use case diagram

- I have seen excessive focus on drawing the tool-supported diagrams of actors and use cases, with various use case relationships; I have even seen people try to indicated use-case dynamic behaviour that way
- If a diagram is better for you than a structured list, then use it; it is, after all, merely an index to the use cases
- Use <includes>, which used to be <uses>, but avoid <extends>
- Create a diagram indicating how the business process uses use cases, to show the sequences in which they may be used
- Create some scenarios to show dynamic behaviour
- Don't draw the diagram until you have several use cases written

Use of use cases when another formalism would be better

- A state machine or activity diagram may be a better bet. In particular, don't try to express complex flows (such as an automated process may use) using a use case template

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 BPM or the calling use case

RUP and the Rose tool

- RUP is tailorable, but inexperienced users overemphasise the use case diagram and need more help in choosing a use case template
- Capturing a BPM in "(textual) business use cases" works poorly as business processes involve many actors over time, for which a diagrammatic notation is much clearer. Activity diagrams can be used, but in Rose they flow top to bottom, which is less clear than the usual left-to-right swimlane notation

Tool support

- There is no good tool support for use case modelling, presumably because there are so many approaches, which, though similar, would require the tool to be very customisable

Link to User Interface

- UI detail must be kept out of the functional spec, but needs to be linked to it. Sometimes overall UI guidelines plus simple annotation of the use case is enough as screen design documentation. This is an area that needs exploration

Object orientation

- Use case modelling is not an OO technique, and poor use case work should not be bolstered up by some analysis object model that created classes and allocated responsibilities in some fanciful way that may or may not be useful in design and development. It is the UCM that the client will see and agree.

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 sometimes I actually use an explicit parameter. I wouldn't call this business 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, but can have a generic structure of 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 creations of an appropriate notation. There is an example in an Appendix

Local/global and the architectural influence

- Above I say that 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 UCM Preamble and to the modification of the UC template and the guidelines. What can we write down about good practice in this area?

How much to say about system internals?

- I started off in use case modelling by banning things like "the system updates the Customer Details", because it details system internals, and requiring a further use case that pulled out the details and found them updated. Now I think this is too fussy. The "system updates ..." statement is shorthand for several use cases; what we are saying is "... and in any use case that now accesses the Customer Details the updated information will be shown". We might have had "... the updated Customer Details are posted to overnight batch update", implying they weren't available until next day. If working with the batch system is a given, then this should be in the preamble, and the effects taken into account. Note that there is nothing about databases here - this is about abstract business state.

Layering the use case model

- I met a man at a conference who talked about a very large use case model for a military system that had a layer of high-level use cases, with each broken down into lower level ones, which were again broken down. I didn't understand this, and he couldn't tell me the (classified) details. Perhaps really they were business processes shown at different levels of granularity, or perhaps they were functional domains. I'd like to have known.
- 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"

Sequence, detail and accuracy over time

- I have found it hard to get people to think of use cases as other than either "finished" or "not finished", but then too often thinking that iteration means "its OK to finish it later". Some criteria for phase completion are given in an Appendix. It can also be useful to name intermediate states, so that for example having the name, goal and pre/post conditions is called "outlined"

Needing a vision

- You can make use cases only for a system that you can imagine already. At the stage where you (as end-user or developer) are wondering what kind of system will fit the bill, they you need to do some envisioning. 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 modelling, as we synthesise the specifications

Conclusions

Use case modelling is a vital technique in system development, and there are many issues of many kinds in developing powerful models. One overall question for this workshop is the extent to which UML can assist; my hope is that it will address many of the issues I have listed.

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 course, and clients of our methods.

Thanks also to the anonymous reviewer of the paper who pointed out the inconsistencies in my thinking about pre/postconditions.

References

Bittner, K and I Spence (2003) Use case modelling. Boston: Addison-Wesley [very clear that modelling is synthesis, and on the need for a vision]

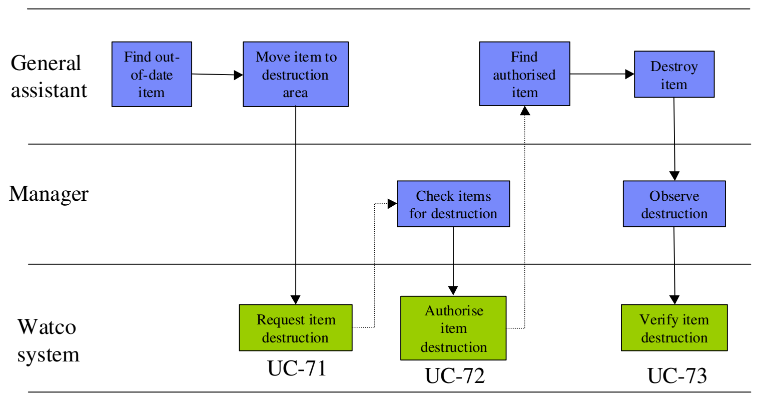
Cockburn, A. (2001). Writing effective use cases. Boston: Addison-Wesley [much of my thinking is derived for Alistair's work]

D'Souza, D and A Wills (1999). Objects, components and frameworks with UML. Boston: Addison-Wesley [business state, pre/post-conditions]
Lewin, K. (1951). Field theory in social science. New York: Harper Row

Appendices

Appendix 1 - Business Process Model tied to Use Cases

BP-200: manage out of date stock



Appendix 2 - IBM GSM criteria for UCM completion by phase (in Custom AD)

Introduction The technique of use case modelling 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 modelling, 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 each release cycle. Here are suggested completion criteria for use cases at the end of Solution Outline and the end of Macro Design respectively.
Actual completion criteria will depend on the nature of your 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

Appendix 3 - Example of generic use case

Main success scenario

 ...
 System shows <u>loan types</u>
 User selects a <u>loan type</u>
 System requests <u>loan application information</u>
 User enters loan <u>application information</u>
 System shows <u>warning notices</u>
 User acknowledges <u>warning notices</u>
 ...

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

Appendix 4 - 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