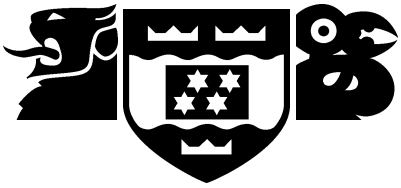
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

Use Case Cards and Roleplay for Object Oriented Development

Robert Biddle, James Noble, Ewan Tempero

Technical Report CS-TR-01/4
March 2001

Abstract

This paper presents a technique involving index cards and roleplay to assist in making use case development more accessible and better guided. The technique is based on the established CRC card technique used for object-oriented design. In our technique, essential use cases are recorded on cards, and roleplay is used for development and review. The paper presents the technique, and outlines our experience in applying it. We found the technique did achieve the expected benefits, and was especially useful in determining the system boundary. We also saw other significant benefits, some relating especially to the nature of essential use cases.

Author Information

The authors are lecturers in Computer Science at Victoria University of Wellington, New Zealand.

Use Case Cards and Roleplay for Object Oriented Development

Robert Biddle, James Noble, Ewan Tempero
School of Mathematical and Computing Sciences
Victoria University of Wellington
Wellington, New Zealand
{robert,kjx,ewan}@mcs.vuw.ac.nz

Abstract

This paper presents a technique involving index cards and roleplay to assist in making use case development more accessible and better guided. The technique is based on the established CRC card technique used for object-oriented design. In our technique, essential use cases are recorded on cards, and roleplay is used for development and review. The paper presents the technique, and outlines our experience in applying it. We found the technique did achieve the expected benefits, and was especially useful in determining the system boundary. We also saw other significant benefits, some relating especially to the nature of essential use cases.

1 Introduction

We have been exploring ways to improve the techniques we use in the early stages of object-oriented development. In particular, we have been investigating how to better determine use cases that capture requirements and can drive design. We were looking for techniques that would ensure active engagement by team members, provide more operational guidance, and also make the techniques more accessible to learners and system stakeholders. This paper presents a technique we have developed using index cards and roleplay.

A use case describes a sequence of interaction between a user and a system. Identification and employment of use cases is now common practice in software development, and the use case is now a recognised concept in modeling languages and in development processes. As with many other aspects of object-oriented analysis, however, use cases require understanding and experience before they begin to deliver their potential. Moreover, these activities need techniques that are lightweight, inclusive, and flexible, so to best ensure insight and agreement about desirable system behaviour.

The paper is organised as follows. In the next section we provide some background, discussing use cases and their role in object-oriented development, and also reviewing the CRC card design technique we adapted to use cases. In section 3 we describe the technique itself, and in section 4 we review our experience. In section 5 we explain our use of *essential* use cases, and the implications. In the final section we present our conclusions.

2 Background

2.1 Use Cases

In his 1992 book[6], Ivar Jacobson defines a use case as "a behaviorally related sequence of transactions in a dialogue with the system". The general idea is to represent intended sequences of interaction between a system, even if not yet implemented, and the world outside that system. This idea is very powerful, for several reasons.

In the early stages of development, use cases help to focus on interactions as a way of eliciting intended or desirable system behaviour and so capture requirements and help determine specifications. This technique is effective because interactions can be described in forms very easy for people to recall or imagine, such as narratives or dialogues.

In the later stages of development, use cases help again because of the focus on interactions. The interactions can now be regarded as the embodiment of specifications that the system must meet. In design and implementation, a structure must be determined and created that will meet these specifications. In review and testing, use cases can be used to drive system behaviour for examination.

Use cases are based on sequences of interaction, and desirable interactions typically follow a structure of coherent progression, on a limited scale, toward a goal or sub-goal.

This allows a useful partitioning of specifications. The partitioning into use cases is helpful in overall management throughout development, because use cases can be organised by selecting, grouping, filtering, prioritising, and so on.

2.2 CRC Cards

When looking for a technique to make use cases more accessible, we were inspired by a technique that addresses a later activity in object-oriented development. This technique is CRC (class-responsibility-collaborator), which uses cards and roleplay to facilitate designing a system as a set of collaborating objects [2, 9, 3]. Each class is represented by an index card, and members of a design team play the roles of classes, verbally simulating the behaviour of the system. The cards themselves are used to record the responsibilities and collaborators of the class. A responsibility is an abstract idea of what the class should know or do, and the notion of responsibility is used as a heuristic to distribute intelligence in the system.

CRC cards were originally written up as a way of learning about design, where the cards helped make the idea of objects more concrete, and the roleplay fostered an awareness of object collaboration. CRC cards are now widely regarded as a sensible design technique in general, and not only something for beginners. Bellin and Suchman Simone refer to CRC as a "meta-cognitive" process, where the operational nature of the technique assists thinking about key issues in the design[3]:

> *Each person on the team literally takes on the role of a class and, using the CRC card as a script, acts out the system. The value of this strategy is that the act of pretending to "be a class" and figuring out what you have to do triggers the same responses as brainstorming. Playing with the cards triggers unanticipated insights. Role play does this successfully because it makes team members active participants.*

3 Use Case Cards and Roleplay

We have used and admired CRC cards for some time, and when looking for a way to make use cases more accessible, we decided to take a similar approach. Our basic idea was to use index cards for use cases, and to somehow involve roleplay as well.

Use cases describe interaction with a system, but there are several ways to describe interaction. We chose the form of a dialogue between a user and the system. We particularly hoped this would facilitate roleplay, because a use case in dialogue form resembles a script. The script has two roles, user and system, so use case roleplay can simply involve two people, each with a part in the script.

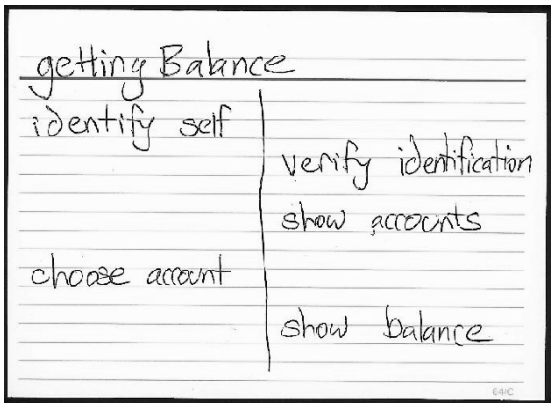


Figure 1. A use case card describing an essential use case for a banking system.

We decided that each use case card should represent a single use case, and should show the name of the use case, and the dialogue script. To easily distinguish the two roles, we split the card with a vertical line down the centre, and write the user's lines on the left, and the system's lines on the right, as shown in figure 1. This layout resembles the two-column format used by Wirfs-Brock [12]. This kind of layout is especially good for functional requirements because it highlights how the system is used, and how the system behaves. Non-functional requirements can also be accommodated, simply by making brief notes at the bottom of the card, or on the reverse side.

Use case cards and roleplay assist primarily in elaborating use case "bodies", determining the steps of the interaction. Before that can happen it is first necessary to identify the use cases. This initially involves background analysis to come up with suggestions for the different ways users may interact with the system. In large system development, even use case identification can constitute an activity of significant size and scope. Our approach is to use analysis techniques to identify a wide variety of candidate use cases, and then prioritise them, selecting a few "focal" use cases that represent critical interactions.

We then start to work on these focal use cases by elaborating their steps with cards and roleplay. We generally suggest doing this with a small team of 3 to 6 people, similar to that recommended for CRC cards. We begin to work with cards by writing brief names for the focal use cases on the cards. For example, a banking system might have focal use cases called *depositing cash*, *checking account balance*, and *withdrawing cash*.

We then select a card and begin to work out the dialogue. We select people for the roles of user and system, and use

Figure 2. A team performing use case roleplay while others review the dialog.

an exploratory kind of roleplay rehearsal. The team works together on determining the steps in the dialogue. When ideas seem reasonable, the roleplayers "play-act" through the script, as shown in figure 2. The rest of the team audit, and afterwards the dialogue is discussed and improved where necessary. Increasing availability of document cameras means cards can easily be projected on a large screen for larger groups to follow. This process is applied iteratively until the team is satisfied the dialogue represents the way the user and the system should interact. Sometimes it helps to leave one use case and work with another, returning later to improve the earlier one.

The primary use of roleplay is for exploratory and iterative development of the use cases. We also use roleplay as a way of presenting use cases for review, however. Such reviews may be conducted by peer developers, by people working on other aspects of a project, or by stakeholders or experts in the system domain.

After the index cards and roleplay have helped determine the use case bodies, the full details may be recorded in requirements documentation or in CASE tool databases. The use cases can be used to drive system design and review, and again later in system testing and demonstration. The cards and roleplay may still prove useful in some later activities, where their immediate and lightweight nature support rapid review and exploration of interaction alternatives.

4 Experience

We have now observed many teams applying use case cards and roleplay. We have used the approach in working with more than twenty teams in industry, where the teams typically included non-technical staff such as business analysts and line managers. We have also introduced the approach in teaching and project mentoring at our university, especially in our undergraduate courses in object-oriented development and software engineering. Our experience has been very positive.

We had explicitly sought active and immediate engagement by team members, and that has happened as expected. As with CRC cards, the concrete element of the index cards, together with the behavioural element of the roleplay, together focus attention and command active participation. This alone is valuable because it ensures a focus on determining requirements that involves the whole team.

The index cards also help in a way that, as with CRC cards, relates to their simple nature as cards: they are discrete and concrete, and only a lightweight investment. Cards can be arranged or prioritised on a table top, needed cards can be selected while leaving others, and cards can held during roleplay. A card can be changed or discarded easily without disturbing others, and the change can made immediately.

The advantages of roleplay go deeper. People, even non-developers, are good at following dialogue. Moreover, they are familiar with the dramatic device of roleplay and the willing suspension of disbelief on which drama depends. People are typically able to watch roleplay, and imagine the interaction with the finished system with real understanding. This is a form of fast prototyping, and it allows fast review and feedback that allows iterative improvement.

Bellin and Suchman Simone point out that CRC roleplay leads to unanticipated insights, and the same thing happens in use case roleplay. In CRC roleplay, the insights typically concern how best to distribute intelligence among a set of collaborating objects. In use case roleplay, the insights relate to how best to communicate across the boundary of the system.

The cards and roleplay technique themselves are not always sufficient in themselves to determine whether use cases are reasonable or not. There are many aspects to use cases, and developers need to be aware of the issues. In particular, developers need to be aware of studies about how use cases model systems [7, 1], and studies of what mades use cases actually useful [4]. Wirfs-Brock [13] discusses how good use cases resemble "meaningful conversations", and this especially relates to our technique. Roleplay makes the conversations come to life, and makes it easier to assess how meaningful they really are.

In our experience, one critical issue in determining requirements is simply determining the boundary of the system, distinguishing what the system should do from what it should not do. It can be very difficult reach the agreement necessary to make such distinctions while involving all the people involved, analysts and stakeholders.

Use case cards and roleplay have proven very useful in determining the system boundary. Our approach is to apply an approach familiar in design: we regard the system as a "black-box". The internal workings are not specified, but the way the system is used is specified by the use cases. The two-column dialog form of each use case card clearly shows

Take 1:

User: I say which performance I want and the system shows me the performance details.

CUT! — it's the system's job to say what the system does. This is often just an error made by the role-player, but can also indicate confusion as to where the system boundary is.

Take 2:

User: I say which performance I want.

System: I display the performance details and say whether or not the seats are available.

CUT! — the seats haven't been specified yet.

Take 3:

User: I say which performance I want.

User pauses waiting for a response, then looks over to the person playing the system, who is still looking at the use case card, and doesn't realise he's being cued.

System?

System: You're supposed to say what seats you want to know about too. *Points at card.*

User: Oh, right

CUT. The roleplay does not allow anyone to hide — all participants have to engage with what the use case is about.

Take 4:

. . .

Figure 3. A sample dialogue from roleplay of a use case in a theatre booking system, illustrating how roleplay leads to early detection of use case difficulties.

the role of the user distinct from the role of the system, and the line dividing these roles is the boundary of the system.

There are several ways in which use case roleplay assists to determine the system boundary. In early exploratory roleplay, it can quickly become clear if team members differ in their understanding about what the system is required to do. People reading the same analysis documents can come up with different interpretations, and it is especially common for differences to arise between technical designers and business analyst or domain experts. It is important to detect and resolve such differences early, and determining actual interaction sequences for roleplay tends to expose these differences.

The exploratory roleplay can also expose unreasonable assumptions about both the user and the system roles. For the user, it can become apparent that actions specified in a use case are inconsistent with how an actual user is likely to behave. For the system, it can become apparent the required behaviour is not possible because critical information will not be available. Figure 3 illustrates anomalies can be detected while roleplaying. Actually play-acting the roles seems to make such anomalies more obvious than if the use case was just read carefully.

To resolve such anomalies, the use case steps may have to be modified, or the use case may need to be structured in a different way. Such changes may require modifications made to other use cases, and may even require even creation of new use cases. The important result is that problems can be identified and fixed at this early point, rather than at later more expensive points in development.

Roleplay also makes use case interaction understandable to observers outside a development team. This is especially useful in presenting use cases to stakeholders. Roleplay is quick, immersive, and very accessible. Without heavy investment, it makes is possible to discuss issues, compare alternatives, and possibly detect flaws.

To support use case roleplay and discussion about the system boundary, we have found use case diagrams helpful. We use a UML [8] use case diagram that shows actors (the UML term for users) as stick figures and their involvement with use cases as ellipses. We also explicitly show the system boundary, depicted as a box surrounding the use cases, with the lines between the actors and the use cases crossing the boundary, as shown in figure 4. The convention of showing the system boundary in use case diagrams was used by Jacobsen [6] but does not typically feature in UML. The depiction of the system boundary is helpful in visualising the system as a unit, and the boundary line is consistent with the line dividing the user and the system on the use case cards.

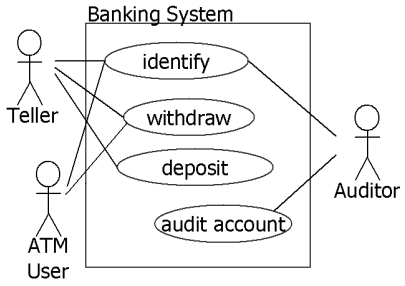


Figure 4. A use case diagram, explicitly showing the boundary around the system object.

5 Essential Use Cases

While we have described the cards and roleplay as involving use cases, we in fact typically work with a refinement of use cases known as *essential use cases*. This refinement was developed by Constantine and Lockwood as part of "Usage-Centered Design" [5], a process for developing user interfaces. They make an important observation:

> *"In particular, conventional use cases typically contain too many built-in assumptions, often hidden or implicit, about the form of the user interface that is yet to be designed."*

Constantine and Lockwood define essential use cases as "a simplified and generalized form of use case". They use the term *essential* because these use cases: "are intended to capture the essence of problems through technology-free, idealized, and abstract descriptions".

The abstraction in an essential use case does not relate to the use case as a whole, but more to the steps of the use case. In this way an essential use case does specify a sequence of interaction, but a sequence with abstract steps. For example, figure 5 shows an ordinary use case for a banking system, detailing specific concrete steps in the dialogue such as insert card and read magnetic stripe. Figure 6 shows an equivalent essential use case that begins with the more abstract "identify self". Also, note that the ordinary use case labels the roles "user action" and "system response", stressing an actual concrete interaction. The essential use case instead takes a more abstract yet also more rich approach, by casting the user part of the dialogue as "intention", and the system part as "responsibility".

We have adopted this form of use case for our cards and roleplay for several reasons. The abstraction keeps essential use cases brief, and so able to fit on an index card. Also, the abstraction helps avoid unnecessary debate about irrelevant implement details, so allowing more rapid progress to be made. The focus on intention and responsibility also has important consequences.

gettingCash

User Action	*System Response*
insert card	
	read magnetic stripe
	request PIN
enter PIN	
	verify PIN
	display transaction menu
press key	
	display account menu
press key	
	prompt for amount
enter amount	
	display amount
press key	
	return card
take card	
	dispense cash
take cash	

Figure 5. An ordinary use case for getting cash from an automatic teller system. (From Constantine and Lockwood.)

Our experience with essential use cases has shown that the simple benefits are realised, and there are also more profound effects. As we had hoped, the abstraction does keep the use cases brief enough for an index card, and also helps avoid premature debate about implementation.

On a deeper level, we have seen there is heuristic merit in casting the user role as expressing "intention", and the system role as expressing "responsibility". Together, these lead to use case dialogues that better reflect the real motivation of the user, and better capture the requirements of the

gettingCash

User Intention	*System Responsibility*
identify self	
	verify identity
	offer choices
choose	
	dispense cash
take cash	

Figure 6. An essential use case for getting cash from an automatic teller system. (From Constantine and Lockwood.)

system, while at the same time avoiding premature design decisions.

We have found there are some adjustments needed in working with essential use cases. One adjustment is that use case roleplay will itself be abstract, and so not as realistic as if it involved specific interface details. In our experience this has not been problematic, although teams do find it helpful to sometimes consider concrete enactments of the use case to check their understanding. Another possible adjustment is that essential use cases could easily accommodate system responsibilities that do not directly involve interaction. For example, the use case in figure 5 might well involve responsibilities to maintain account balance integrity and to audit trail entries.

The focus on the system part of the dialogue as documenting responsibilities has interesting and far-reaching implications for system development. When use cases have been determined, one of the later activities in object-oriented development is the design of the system as a set of collaborating classes and objects. One approach to determining this set is to focus on responsibilities: this is a key part of using CRC cards, and is used more thoroughly in responsibility-driven design [10, 11].

Responsibility is good heuristic for designing objects that involve behaviour and date working together, because the word "responsibility" suggests both a duty to do something, and the resources with which to do it. Responsibility also allows delegation, allowing large responsibilities to be managed by delegating smaller responsibilities to others.

Essential use cases capture the system role as a set of responsibilities. This means that when all the use cases are determined, we also have the set of responsibilities for the whole system. We can regard the system as a single object, and the responsibilities as the specification for the object. We can design as set of collaborating objects as a decomposition of the system object, with the system responsibilities distributed appropriately.

Techniques such as CRC and responsibility-driven design already address how to distribute responsibility. But these techniques begin with responsibilities associated with objects and classes identified using domain analysis. Essential use cases deliver a set of responsibilities specific to the system being designed, which presents valuable opportunities.

Ultimately, the system responsibilities and the object set responsibilities must match. There are two major implications. Firstly, we can begin CRC or responsibility-driven design with responsibilities informed by the essential use case responsibilities, as well as the results of domain analysis. Secondly, we can check the set of object responsibilities at any time to see if they indeed will meet the system requirements. Even if we rapidly explore design alternatives, or consider existing reusable design assets such as patterns, frameworks, or components, we will always be able to check that the objects work together to fulfill the responsibilities specified by the use cases. The first implication means better operational guidance in beginning design. The second implication means better traceability between design and requirements.

6 Conclusions

We have presented our technique involving index cards and roleplay to help determine use cases. We were looking for ways to make use case development more active, accessible, and better guided. We based our technique on the established CRC card technique used for object-oriented design.

Our technique is lightweight and low in formality, but it is easily employed and in our experience has been very effective. We did get the benefits we set out to achieve: the technique gets people involved rapidly, and guides use case development in a sensible way. We also found the techniques were effective in exposing potential flaws and anomalies in use cases, and was especially helpful in determining the system boundary.

Other benefits related to our adoption of essential use cases. These use abstraction in their steps by emphasising user intention and system responsibility. We found these characteristics worked well with our use of index cards and roleplay, and made use case analysis both easier and more powerful. In particular, the identification of system responsibilities provides opportunities to better guide object-oriented design, and leads to better traceability between design and requirements.

References

[1] F. Armour and G. Miller. *Advanced Use Case Modeling: Software Systems, Volume 1*. Addison-Wesley, 2001.

[2] K. Beck and W. Cunningham. A laboratory for teaching object-oriented thinking. In *Proc. of OOPSLA-89: ACM Conference on Object-Oriented Programming Systems Languages and Applications*, pages 1–6, 1989.

[3] D. Bellin and S. Suchman Simone. *The CRC Card Book*. Addison-Wesley, 1997.

[4] A. Cockburn. *Writing effective use cases*. Addison-Wesley, 2001.

[5] L. L. Constantine and L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage Centered Design*. Addison-Wesley, 1999.

[6] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering*. Addison-Wesley, 1992.

[7] D. Rosenberg and K. Scott. *Use case driven object modeling with UML: A practical approach*. Addison-Wesley, 1999.

[8] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[9] N. Wilkinson. *Using CRC Cards - An Informal Approach to OO Development*. Cambridge University Press, 1996.

[10] R. Wirfs-Brock and B. Wilkerson. Object-oriented design: A responsibility-driven approach. In N. Meyrowitz, editor, *Proc. of OOPSLA-89: ACM Conference on Object-Oriented Programming Systems Languages and Applications*, pages 71–75, 1989.

[11] R. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object Oriented Software*. Prentice Hall, 1990.

[12] R. J. Wirfs-Brock. Designing scenarios: Making the case for a use case framework. *The Smalltalk Report*, 3(3), 1993.

[13] R. J. Wirfs-Brock. The art of meaningful conversations. *The Smalltalk Report*, 4(5), 1994.