AP-M Appropriate Process Movement

Effective Modelling with Use Case Chains

Hüseyin Angay

Karabash Ltd.

Member of the Appropriate Process Movement

http://www.aptprocess.com

Abstract

Use case modelling requires a constant balancing of the trade-offs between the complexity of the networks of use cases and the complexity of the contents of the use cases. We can reduce some complexity in the use case text at the expense of increased complexity at use case relationship level. Or, we can simplify use case relationships at the expense of increasingly complex use case text. *Use case chaining* is a technique for reducing the negative impact of complex use case networks. We can model with relatively large numbers of use cases and using simpler use case text, but we can reduce the network's complexity whenever we need it. The technique allows teams to work with use cases in a more agile manner, without worrying about complexity trade-offs but neither sacrificing the quality of the model. The paper gives a number of examples where the technique may be applied, including requirements elicitation, elaboration, verification and project scheduling.

Copyright Note

This document resides online at http://www.aptprocess.com/whitepapers/ and has been authored by Hüseyin Angay of Karabash Ltd. and the Appropriate Process Movement. It may be copied freely in part or in whole, with the restriction that anywhere using a copy of more than three paragraphs must include as reference the web address of its origin, as given above.

Copyright 2003, Appropriate Process Group

Contents

Abstract 1
Copyright Note 1
Contents 2
Introduction 3
Cramped use case styles 3
What is a use case chain? 6
Putting the chains to use 9
  Model validation 9
  Model walkthroughs 9
  Feature cataloguing 10
  Prioritization 11
  Scheduling development based on resource availability or risk 13
  A more elaborate scheduling strategy 14
  Test scenarios 18
  Verification and elaboration of Grand Scenarios 19
    Example 19
Further work on the subject 24
  Automation 24
  Pre- and Post-conditions 24
Summary 25
Acknowledgements and References 26
  References 26

Copyright 2003, Appropriate Process Group

Introduction

Recent trend in use case modelling is towards larger use cases with only a few relationships with others. One of the reasons for this trend is manageability: As the use case models become more elaborate, they become harder to manage. Unfortunately, as the use case size increases, some of the advantages of use case modelling are lost since they ultimately end up pretty much like the requirements specification documents they were supposed to have replaced. Here, I propose a method for structuring a use case model that can help to exploit the advantages of smaller use cases without losing the manageability.

Cramped use case styles

So, why should we aim for smaller use cases?

First of all, let's establish what we mean by large or small. A use case that has three or four alternative flows is small; so is one with ten or twenty steps or those structured in such a way that the textual content does not exceed a couple of pages. The main criterion, however, is that the functionality described is significant but not too complex – i.e. you cannot see more than about a six or seven scenarios/paths through one use case.

A large use case is one that has more than six or seven alternative flows or one whose textual content runs into several pages. One sure sign that the use cases are erring towards the larger end is that the modellers feel the need to describe almost every use case with activity diagrams. If the text starts to get so complex that the activity diagrams are inevitable, more often than not, the modellers will not have been making use of the techniques for reducing complexity but will have aimed to keep the pictorial model small in order to satisfy some model complexity heuristic.

One aspect that is often misunderstood about use cases is the nature of the relationships between them. In UML, as the relationships between classes and those between use cases share their nomenclature and as the icons are close enough graphically, there is a tendency to treat both types of relationships in the same way. So, the relationship arcs between use cases become communication routes. In the same manner, the generalisation of use cases is visualised as if classes were involved, to the

Copyright 2003, Appropriate Process Group

point that modellers start to ask what one use case may be inheriting from another one. There is a much simpler way of looking at relationships in use case models: They are text management artefacts. Don't interpret this as a dismissal or denigration of these relationships. I think they are quite crucial to the effectiveness of use case modelling, but their utility should be measured in different terms from the utility of relationships in class models. Therefore, their heuristics will be different, too.

Having established that one of the main characteristics of use cases is text management, what features make them more useful?

- Reducing repetition
- Reducing complexity
- Ability to treat them as atomic units of functionality

Unfortunately, the aims of reducing repetition and complexity conflict with that of treating all use cases as atomic units. The trade-off between complexity of the contents of individual use cases and the complexity of the model (that is, the system of use cases, actors and their relationships) is something we have to live with. Whichever one we favour, we will violate the other. We can reduce a use case’s text to its bare essentials and play all the style tricks available to us to make it as concise as we can, but there is a minimum amount of text that we will need to capture and maintain. If we reduce it any further, we will lose essential detail. When we reach that point of minimal text, the partitioning strategy that we adopt will either reduce the complexity of the text in the use cases at the expense of increasing the complexity of the model or vice versa.

The trade-off between these two complexities is a matter for personal choice, but that choice is hopefully based on the needs of the project or the enterprise, rather than on some purely subjective criteria.

The factors that should influence an objective decision are:

- **Communication:** The more complex the model or the text of the use case, the harder it is to read and to follow. Complexity becomes a source of misunderstandings and frustration.
- **Elicitation:** The more complex the model or the text, the harder it is to spot omissions and inconsistencies.

Copyright 2003, Appropriate Process Group

- **Control:** Complexity affects the physical control of the model. Documents and other model files need to be moved around, printed, versioned, baselined and shared amongst team members. Configuration management can become a big issue in large projects.
- **Manageability:** Use cases are the main work units in enterprises that adopt them fully. Atomic units are much more easily assigned, used and tracked than units that are shared between teams, iterations and releases.

Depending on the enterprise and, in fact, the project involved, the appropriate trade-off may veer towards more complex text or a more complex model. My experience is that we have some flexibility to err in either direction to maximise communication, elicitation and control, whereas manageability aspects almost always point to smaller use cases in all but the most trivial of projects. The "80% complete" problem is endemic enough that I would always advise that one use case's implementation should not be split between two iterations, for instance. This alone should encourage teams to adopt smaller use cases[1]. Yet, I have been in enough projects where a complex model caused problems in the first three factors to know that going for smaller use cases is not always sensible or popular.

The technique described in this paper and the associated tools are intended to reduce the impact of the complexity of the model in order to make it easier to work with smaller use cases. The property of atomicity is often ascribed to use cases. However, the word *atomic* carries different meanings according to the circumstances. It can refer to atomicity of a transaction, i.e. a single meaningful interaction from the beginning to the end; or it can refer to the atomicity of a section of text, i.e. we can take this section of text away from the a body of text, but dividing it any further makes no sense. The use cases in a single model will rarely ever achieve

[1] There is a school of thought that equates small use cases with functional decomposition, the bête noir of the use case scene. This fear is not baseless: One of the symptoms of functional decomposition is a large number of small use cases. But we should not look at a single symptom and pass a diagnosis – hypochondriacs rarely make good doctors. Functional decomposition applies a partitioning strategy that says "Use Case 4 follows Use Case 3 follows Use Case 2 and so on". The partitioning strategy I recommend is based on factoring out significant functionality and replacing it with placeholders. The aim is to reduce complexity ("We have too many decision points in one use case. It's becoming unreadable. Can we improve it by splitting it into smaller use cases?") and maintenance effort ("We have this shared behaviour in several places. Can we factor it out?").

Copyright 2003, Appropriate Process Group

both designations. Use case chains consolidate the model by reducing groups of use cases into larger but transactionally atomic units at crucial points of the project when that kind of atomicity is useful. In the physical model, however, use cases are still kept as discrete but interrelated units that can be individually managed, controlled and maintained – i.e. atomicity is at the level of textual unit. Keeping the atomicity relevant at all levels, we avoid some of the disadvantages associated with choosing one style of use case rather than another.

What is a use case chain?

A chain (i.e. a composite use case) covers an interaction from beginning to end. It is formed by applying a few simple rules to a number of interrelated use cases. This composite use case can be used for communication, elicitation and planning/scheduling purposes.

The chaining rules are:

(1) If use case A includes use case B, we can form a single use case C where the flow is $A_1 \Rightarrow B \rightarrow A_2$.[2]

(1a) If an alternative scenario of use case A includes use case B, we can form two separate use cases: C = A and C'= A+B. (The inclusion is conditional, but this fact is hidden within the use case text. It is debatable if this is good modelling style. If at all possible, I would encourage the modeller to use an «extend» relationship instead.)

(2) If use case D extends use case E, we can form two use cases:

1. Use case F where the flow is $E_1 \rightarrow D \rightarrow E_2$;
2. Use case E on its own.

(3) If use case G is a specialisation of use case H, we can form a simple use case I by combining the descriptions from G and H.

(4) A generic use case never appears alone. It is always paired with one of its specialisations. In other words, we consider generic use cases to be abstract use cases.

(5) The first link of a chain is always concrete – i.e. it is not an included or an extending use case.

(6) The last link of a chain does not include another use case.

By applying these rules, we can combine the use cases in the model and form the chains. As long as the use case model is consistent, the rules

[2] A_1, A_2, etc. refer to the segments of the same use case. So, $A = A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_x$.

Copyright 2003, Appropriate Process Group

guarantee that a use case chain captures a complete interaction from start to end.

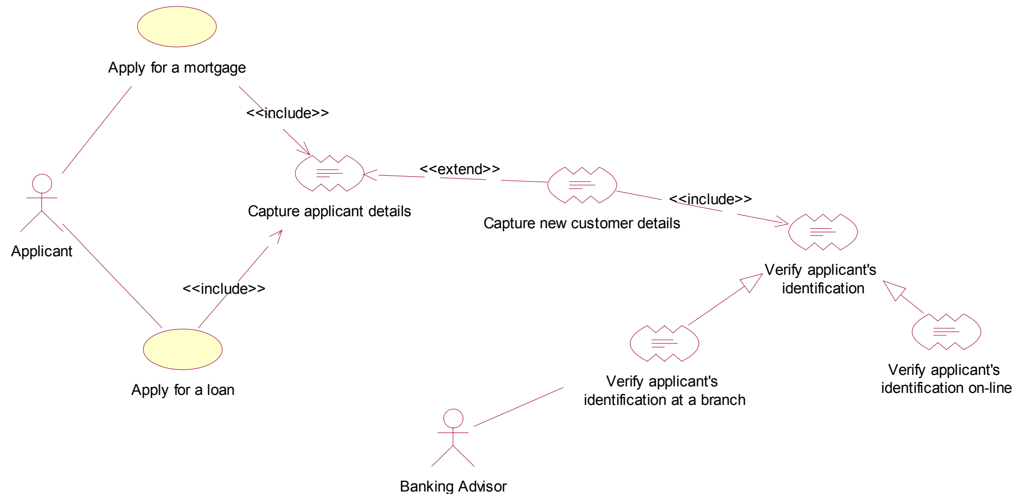

Figure 1 – example use case diagram[3]

Figure 1 shows a simple group of use cases. For the sake of the example, we make the assumption that the two use cases that describe applying for a mortgage and applying for a loan have very little in common, to the extent that they warrant separate use cases. We will elaborate on this model later in the paper.

---

[3] This use case diagram may look strange at first. What are those bits that look like scraps of paper or broken eggs?

I have been applying the concept of chains to use case modelling for some years now and this paper has been written and scrapped in several guises over the last six years. The partitioning strategy I recommend may seem alien to many practitioners who are familiar with one-liner rules of thumb like, "Even a very complex system should have no more than 40 use cases," or anecdotes like, "I had a client who had 150 use cases in their model and they were clearly in trouble at that point." Most of these statements miss the point: While some use cases in the model are the *concrete use cases* that are the framework for the model, the rest are the *use case fragments* which describe partitioned behaviour. The partitioning strategy dictates the nature of these fragments, but there will generally be many more fragments than concrete use cases in the model. The heuristics that demand that the number of use cases be small apply to these concrete use cases. There is a lot more latitude in the way we model the fragments. The partitioning tradeoffs we are talking about are mostly to do with these fragments.

Ivar Jacobson's recent comments on use cases clarify this issue even further [Jacobson 2003]. You can find the icons for use case fragments ready to use with Rational Rose in http://www.aptprocess.com/resources/.

Copyright 2003, Appropriate Process Group

Given this group of use cases, we would have the following chains:

Chain 1:

{Apply for a mortgage} → {Capture applicant details}

Chain 2:

{Apply for a mortgage} → {Capture applicant details} → {Capture new customer details} → {Verify applicant's identification on-line}

Chain 3:

{Apply for a mortgage} → {Capture applicant details} → {Capture new customer details} → {Verify applicant's identification at a branch}

Chain 4:

{Apply for a loan} → {Capture applicant details}

Chain 5:

{Apply for a loan} → {Capture applicant details} → {Capture new customer details} → {Verify applicant's identification on-line}

Chain 6:

{Apply for a loan} → {Capture applicant details} → {Capture new customer details} → {Verify applicant's identification at a branch}

The arrows we used in the chains do not indicate sequencing but levels of embedding. Think of them as «include» relationships.

For specialisations, we used a shorthand and listed only the specialised use case. We left out the generic use case as it is always there by implication.

It's worth noting that even with a contrived and simple example like this (for instance, we left out other extensions possible for details capture), we will have quite a few use case chains. This is not a failure of the technique, but the nature of the systems that we work with: Even simpler systems hide significant complexity. Splitting the model into these small units allows us to manage that complexity when we are building up the model, but we would like to reveal the depths hidden in

Copyright 2003, Appropriate Process Group

that apparent simplicity. Chaining the use cases together like this gives us that capability.

Putting the chains to use

Use case chains have several uses. Some of these are listed here.

Model validation

Some of the heuristics for use case models apply to groups of use cases, i.e. chains. Here are examples of these heuristics:

- *A chain must have at least one actor.* Although not an absolute rule, a chain without an actor is an indication that there may be something odd going on. This heuristic is often applied to individual use cases, but we then have to make exceptions such as, "… unless it is an included use case, etc." Phrasing it within the context of a chain captures the actual intention of the heuristic better.

In fact, most heuristics of use case models apply to chains, rather than individual use cases or use case fragments. Some other rules in this category are:

- *A chain must have an external stimulus to trigger it.* When we apply this rule to individual use cases, it is often hard work to find triggers for use case fragments, so we may resort to dreaming up artificial ones in order to please the process. It certainly makes much more sense to insist for a trigger in a chain.

Model walkthroughs

The list of all the possible chains in a model is also the list of all the *grand scenarios* in the model – i.e. all the different paths that we can take through the model. A *grand scenario* is a single path through a group of related use cases – i.e. one chain.

Note that we will have some difficulty with the hidden optional inclusions, that is, where a use case is included in another's alternative flows. There should really be two chains in this case but looking at the diagram only, it appears that the included use case describes compulsory behaviour.

The grand scenario approach is useful for revealing scenarios that have been missed. It is easy to concentrate on the stand-alone use cases and their immediate neighbourhood and to forget the larger network. Modelling with large use cases might be adopted to reduce this possibility, but this advice carries its own baggage of problems in terms of internal

Copyright 2003, Appropriate Process Group

complexity: There are several complex paths through large use cases, which may lead to hidden optional behaviour as described earlier. A use case chain, on the other hand, is a relatively simple path – unless the use cases involved are very complex; but this is unlikely as the technique encourages smaller and less complex use cases. It is generally easier to automatically generate a use case chain than to analyse the text of a use case for possible paths, especially since the former can be automated much more readily than the latter.

Example:

Imagine the scenario where the system helps a sales operator to take a phone order for a new customer, that is, one without an account with the company. At the last moment, the system refuses to accept either the order (because there is no account) or the attempts to create a new account and to assign the order to that account (because an order must somehow be committed before it can be assigned, but it cannot be committed without an initial account to be assigned to). A typical chicken and egg situation. This is not as farfetched as it appears, but it's not always easy to spot[4]. By looking at the grand scenarios, we could actually see that there was no way to create customers after an order.

Feature cataloguing

The catalogue of all the use case chains in the model gives us a list of all the possible features of the system: One chain represents a single stand-alone feature.

Not only does this help ensure that all the features have been covered (not always easy in a complex system), but it also reveals whether undesirable features crept into the system. The latter assertion may raise a few eyebrows as most of us are used to selectively dropping wanted features to meet deadlines, not to providing more features than

---

[4] This was a real situation. Once an order was created, it could not be assigned to a customer created after the order, because the system showed only the customers that existed when the order was created or opened; the list was not refreshed. So, the order had to be suspended and reopened… but an order not assigned to a customer could not be suspended because it would have caused a database integrity error. For the curious reader, here is the call centre staff's simple solution to this impasse: they had a dummy account that they could use to commit an order; they then created the new customer's account and re-assigned the order to that new account. Before anyone asks why they even had the facility to reassign orders between customers, here is the answer to that, too: This was a wholesaler and many of the customers ordered the same things with small variations. It was much easier to copy or reassign ready-made orders than to enter them from scratch. Finally, when someone listened to what the users wanted, instead of deciding for them what they should want, a new solution could be found: An order templating facility, so that they never had to create orders from scratch again

Copyright 2003, Appropriate Process Group

the users wanted. Yet, think of the number of web sites that will let you submit the same order several times because you clicked refresh or a form button at some wrong point. Many sites now carry a user warning that we shouldn't press the order submitting button again whilst waiting for the page to load. The explanation for the fault is always technical: a second click or a refresh will resubmit the form. My explanation is that order taking has been divorced from order completion to the point that they became separate transactions. Considering a chain as a whole, rather than concentrating on single use cases would reveal the problem and suggest a better workaround than blaming the users for their impatience: How about keeping track of the orders and making sure that they do not get submitted again? This may run against the trend for stateless web server architectures, but if the architecture cannot meet all the requirements, what good is it?

Prioritization

The chains can be ordered by priority or value to reveal the most important implementation units in the model. Again, this concentrates on delivering complete features, rather than stand-alone use cases. We may achieve the same aim by documenting a complete feature as a single use case (i.e. the large grain approach to use case modelling), but we then lose all the text management advantages of having smaller use cases.

When rating use cases, I prefer an assigned *value* rather than a *priority*. Although the value and priority for a use case are often related, I find that talking of value encourages discussions about what some functionality will do for the users and the enterprise, whereas talking of priority encourages a sense of urgency without necessarily considering the reasons behind that urgency. How often have you had a conversation like this?

> "So, which features are most important to you."
>
> "All of them."
>
> "OK. Let's prioritise them individually, shall we?"
>
> Ten minutes later, we have ten use cases marked essential, with another three marked important.
>
> "OK. Shall we now assign these some value? Like, how much do you stand to make or lose if we don't have these features. Let's start with this one for printing letters. It's rated 'important'."
>
> "Oh yes, that's a regulatory one. We can't even launch the product unless we can send those letters to the customer."

Copyright 2003, Appropriate Process Group

> Half an hour later, we have three use cases with a few million against them and ten with tens and hundreds of thousands.

So, I like giving a nominal value of, say, 1 to 10 to a use case. I have also seen value expressed in potential earnings to a company in the business of, for instance, delivering financial products on-line. Naturally, concrete figures are preferable when we can get them. In their absence, a rating relative to each other often works well. The important thing is to find out the overriding goal for the proposed system. Is it making more money or cutting some costs? Is it meeting some regulatory deadlines? Is it to reduce the number of fatal accidents or to improve the quality of a product? Then, choose the appropriate value rating for this goal.

How do we work out the value of a chain?

First, assign values to use cases individually. Consider how important each use case is to the enterprise. Do this even for the use case fragments (say, a use case for taking a single line in an order). Then, use one of the following methods. The selection of the method depends on the nature of the use cases and on the way we assigned the values[5].

- **Simple method:** The value of the chain is the highest value amongst its component use cases. This assumes that a feature cannot deliver more value than any one of its components. For instance, "We can manage 50% more flights if we automate the allocation of slots and 70% more if we allow the operator to intervene during the placement of multiple flights competing for the same slot." In this case, the best value we can expect is a 70%

---

[5] Assignment of value to use cases is a contentious subject, but it is no different from assigning a value to requirements formulated in other ways. The important thing is to capture what a use case will earn for the business or what it will save the business. This may be an increase in sales, the ability to carry out higher volumes of work or a reduction in some cost.

There are also some use cases whose only purpose in life is to meet legislative or regulatory constraints. What about those? The business rationale may dictate that we calculate the cost of failing to meet the regulations – i.e. the fines that may be imposed. (This rationale may of course encourage companies to accept the fines imposed for polluting, say, a river because it would cost even more than the fines to fix a faulty installation, but…). In short, the value placed on a use case must reflect the values of the enterprise in order to ensure a more consistent approach to development. Using value systems that run counter to the enterprise, either through vacuous policy statements ("We value customer service above all." *So, how do you measure customer satisfaction or dissatisfaction?*) or wishful thinking ("We will enforce limits for the amount of discount our salespeople can offer." *So, you're going to sack someone who makes a sale but exceeds their discount quota by 1 %?*) result in systems that meet nobody's requirements.

Copyright 2003, Appropriate Process Group

improvement for the chain that involves both automated flight allocation and operator intervention.

- **Complex method:** The value of the chain is a sum or weighted combination of the values of its component use cases. In cases where the values are combinatorial, this method is more suitable. For example, ability to offer items for sale on-line may be worth £3,000,000 yearly and the ability to offer discounts for bundles may be worth another £1,000,000. The chain that involves the use cases for both simple and bundled orders is now worth £4,000,000. I have also seen use cases valued in terms of values proportional to the company's other business – e.g. 10% increase in market share, in which case a simple addition may not be suitable unless we want to show that we can grab 120% of the market by implementing a dozen similar use cases. I find that there are no universal answers to this question, but that most stakeholders are quite quick in finding an appropriate way of combining or weighting the values.

Scheduling development based on resource availability or risk

The chains can be ordered by risk or by the amount of effort in order to chart a lower-risk or a lower-cost development path. The former strategy helps clear up most risks as early as possible, with the expectation that we want the complete system developed eventually – it is a long-term cost reduction strategy. The latter tries to get parts of the system delivered quickly – based on the assumption that anything that works is better than nothing, even if some development, hence costs, in future may suffer from the initial architectural decisions.

The effort required to develop a use case chain is normally the sum of the effort for its component use cases. With risk, a simple addition of risk levels for each use case is typically enough, too. Remember that we are not talking about absolute precision when we talk about value or risk. Effort estimates may be more precise, but they are still estimates. With this technique, we are trying to capture a feel for which functionality to tackle first based on imprecise estimates. The value of the technique is not in getting perfect estimates, but in working out relative magnitudes in order to be able to say, "This functionality's somehow more valuable than that one," or "This one looks a lot more urgent than that one." It is also a very good way of verifying the intuition we tend to exercise in task prioritisation – the estimates may confirm our intuition or they may show a discrepancy, which would warn us to revise our assumptions. If we are consistent with our assignment of risk, we should have roughly applicable numbers.

Copyright 2003, Appropriate Process Group

One thing is worth of note here: Although we order the chains by effort or risk, the chains that appear later in the list are likely to share use case fragments with chains higher up the list. So, the simple act of ordering the chains affects the effort and risk assessments of a chain: If we have already implemented a use case, the effort required for another chain that contains that use case is going to be lower; neither will we have the same level of risk associated with it.

A more elaborate scheduling strategy

A matrix of effort or risk against a chain's value is an even better way of revealing the appropriate implementation paths.

Figure 2 shows a plot of the values of the chains in a model against their development effort. Each of the stars is a use case chain. It is easy to see that the top left quadrant has the high-value, low-effort chains, whereas the bottom right quadrant has the chains of relatively low value that still require significant effort. Above he green dotted line (top), the value of a use case chain is *probably* more than the effort that will go into implementing it. We marked those green. Below the red dotted line, the values of the use case chain is *probably* much less than the effort that will go into implementing it. We marked those red. It goes without saying that we would normally aim to work with the green use case chains first and leave the red ones until last.

Copyright 2003, Appropriate Process Group

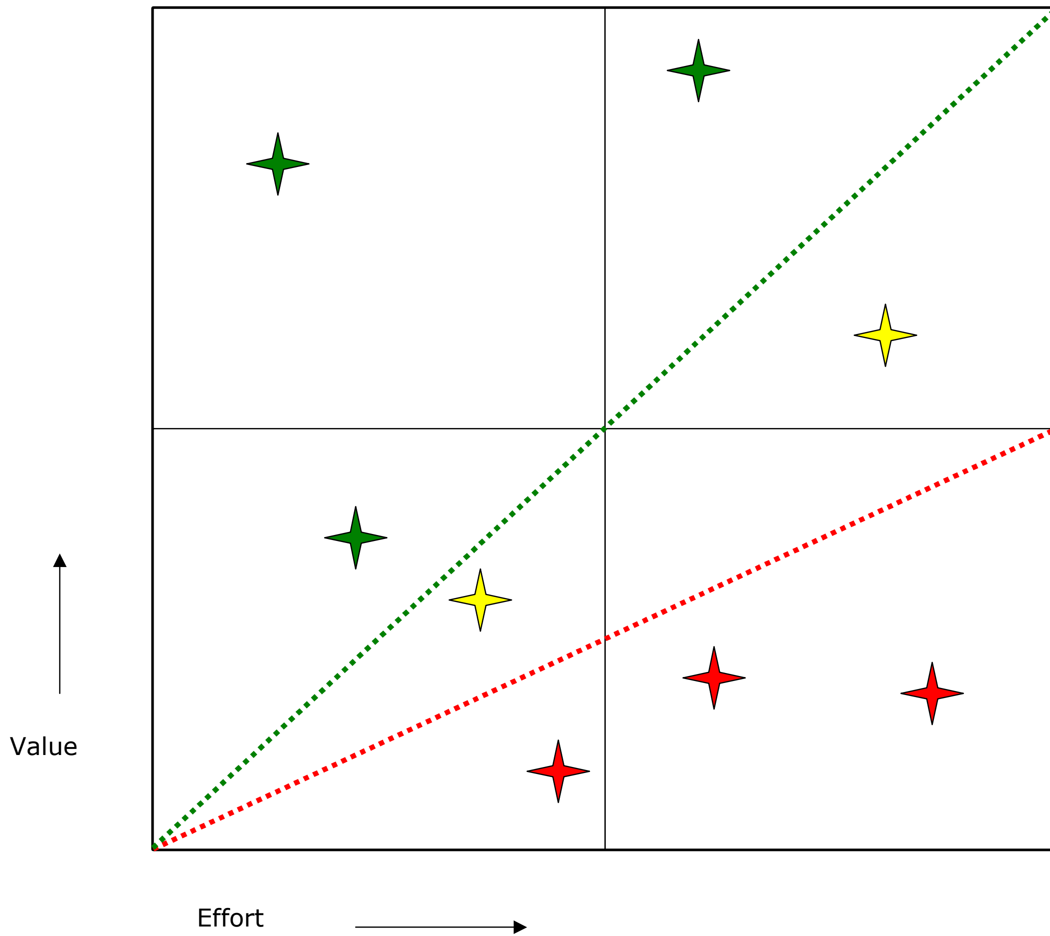

Figure 2 – Effort vs. Value matrix

The same approach also works when comparing the risk with the value of a use case chain. See Figure 3 for an illustration of this.

Copyright 2003, Appropriate Process Group

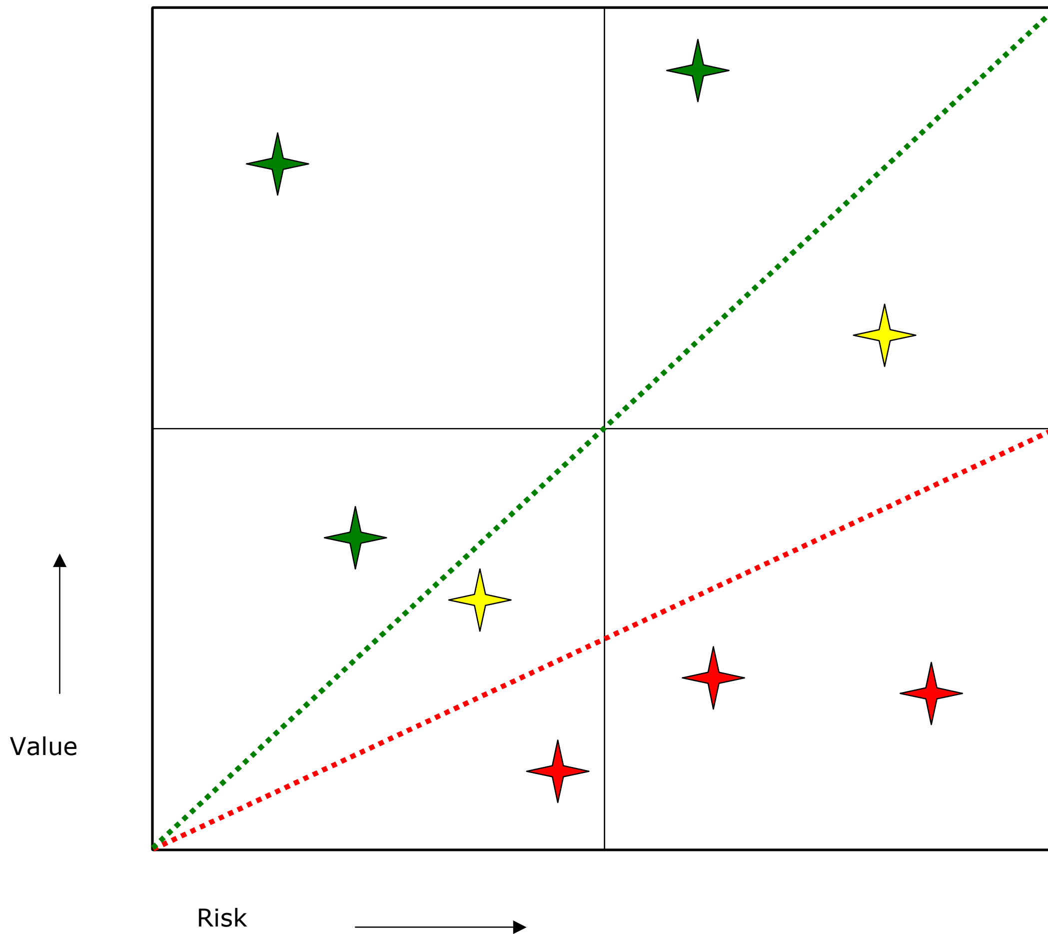

Figure 3 – Risk vs. Value matrix

The approach to risk-based scheduling may differ depending on the enterprise and project. We may, for instance, choose to take on the top right quadrant before anything else just to clear out some significant risk, if there is also value. This assumes that the risk taken at the beginning is architectural risk, that is, it will affect the remaining use case chains, too. It may also be that the system is worthless without the use cases involved, in which case, it's likely that those chains at the top right corner will show significantly higher value than the others, anyway. One-off risks that will not solve anything further down the development path are rarely justified, on the other hand, especially if they are technical risks: there is likely to be a packaged solution that will work in another year's time.

These graphs work on the assumption that we can divorce effort and risk in our assessment of the cost of a chain against its value. This is rarely the case. A more elaborate solution would be to combine the three metrics in a single three dimensional graph. While this may be of

Copyright 2003, Appropriate Process Group

academic interest, a more pragmatic approach is to keep the format of the effort/value diagram but to plot with horizontal segments that combine effort and risk – the segment stretches longer as the risk goes up. In fact, the segment stretches from the least amount of effort expected to the most.

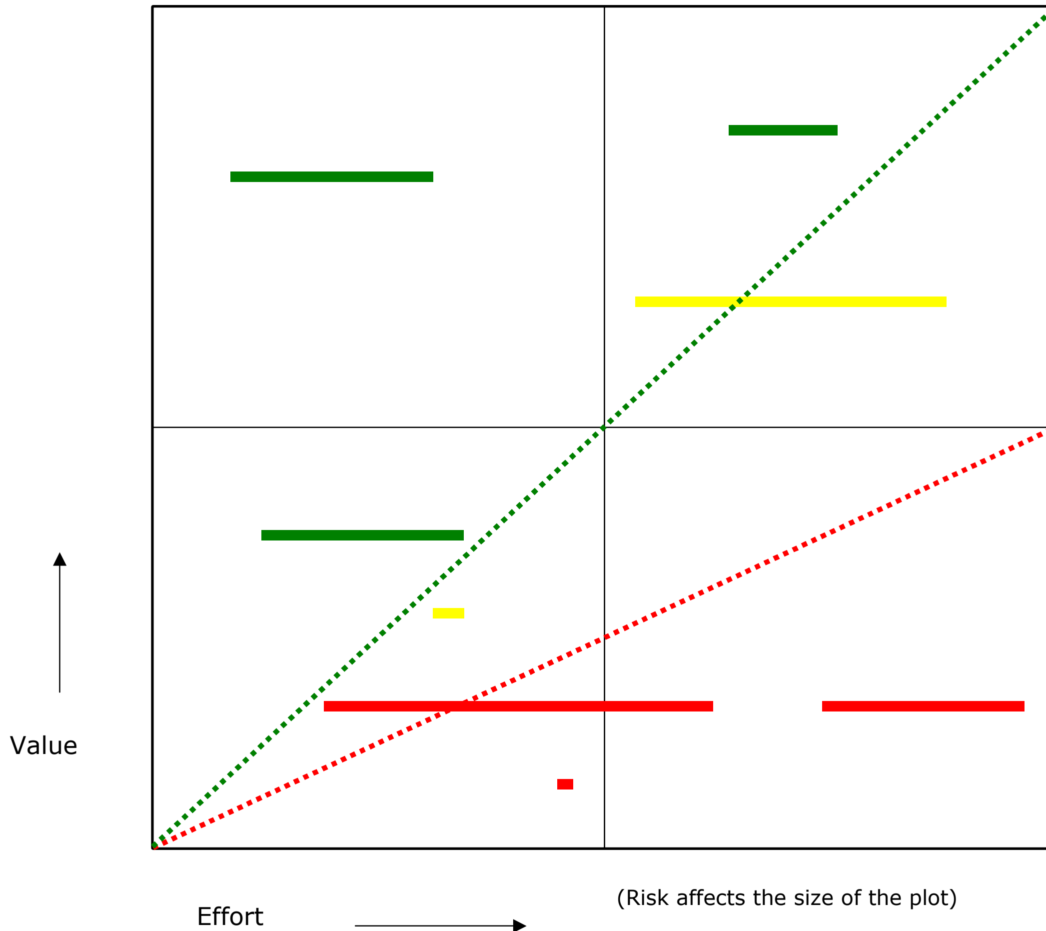

Figure 4 – Effort and Risk vs. Value matrix

This technique is most valuable in the borderline cases. The graph we have here is how a risk-averse project would have interpreted the chains: If the effort risks going into another zone, the chain is given the worst rating possible. Here, the chain that overlaps the green and yellow zones is coloured yellow and the one that overlaps the yellow and the red zones is coloured red. A project that is less averse to risk may take the approach of assigning the chain to the best rating possible.

The likelier approach is a mixture of both depending on the nature of the risk and the phase of the project or programme. Establishing

Copyright 2003, Appropriate Process Group

architectural foundations may need a less risk averse approach than continuous delivery of functionality or products, for instance. Remember that this is a decision support technique: It does not make the decision for us, but offers a condensed view of the problem to help us make a better and quicker decision.

I mentioned the effect that sequencing has on the effort and risk assessments of the chains. For a start, as iterations carry on and we have a better idea of our parameters such as productivity, magnitude of tasks and the details of the technology and requirements, we revise the risk, effort and value assessments. So the shape of the matrix, hence the trade-offs change. More importantly, though, the use cases that have already been implemented can be reduced in effort and risk to near zero – not quite zero as there will be some integration effort depending on the architecture and the development environment. So, the position of a chain in the matrix can significantly change depending on the chains that have been implemented before it. Even the value of a chain may be affected, if some of its component use cases have already delivered their expected value in market share terms, for instance. Working out the trade-offs in this case can get quite, shall we say, interesting. It should be simpler for a computer to work out a development path based on delivered value, time and resources available and the approach to risk. A more likely approach is to let the computer quickly work out a series of candidate development profiles from which the team will choose one.

Again, I must put the stress on the *decision support* aspect. The aim is not to work out a rigid schedule based on a few numbers that may not reflect many of the qualitative factors that will affect the decision. The benefits of the technique come from being able to reduce the amount of effort that goes into working out the schedules. If it takes three days to draw up an iteration plan, a project manager will be very averse to changing the plans once they have been prepared. If it takes half an hour to draw up the same plan, the same manager will be a lot more willing to revise the plans. As the information that drives the schedule can be harvested from the use case model, which should always be up-to-date, the technique allows us to revise our plans quickly and with increasing certainty. This should encourage teams to take a more agile approach to their scheduling and should motivate us to make schedule changes not because of a panic condition but as a matter of fact – in fact, whenever we know more about the situation.

Test scenarios

The set of chains in the model can be used by testers to generate the end-to-end test scenarios. Smaller use cases allow testers to construct

Copyright 2003, Appropriate Process Group

reusable sets of tests much more effectively than working with large scenarios that may have many hidden paths and commonalities. If the test cases are kept as close to the use case model as possible, the use case chains will also reflect the chains of test cases. So, the concept of chains is applicable to both the use case and test case models.

Verification and elaboration of Grand Scenarios

A chain is a complete Grand Scenario. It is, for all intents and purposes, a single use case composed of use cases (both concrete use cases and fragments) from the model. So, when we want a detailed end-to-end description of a process, we can combine the text of all the use cases in a chain to produce a large, all encompassing use case.

This may sound like a waste of time. After all, why split the model into smaller use cases if we are going to recombine them? Remember that the aim of chaining use cases is to allow us to be able to work with smaller use cases when they are appropriate and larger ones at other times. In a workshop environment, end-to-end description of an interaction is much more useful than a set of small use cases. The example that we will be walking through illustrates how we can apply the Grand Scenario technique for the elaboration and verification of a segment of the use case model.

To make this work, we need an automated way to turn a chain into one large use case. Doing the task manually takes too much effort for a throwaway artefact. Inclusion and extension should cause no problems for automation: The text from related use cases can be inserted into the main body at the points where they are mentioned. Unless the writers follow some very stringent writing rules, however, generalisation may require manual intervention.

A single use case also appears to carry some advantages in terms of describing what needs to be implemented, although the disadvantages (we can't track which use cases within the chain have been implemented if we work from a monolithic use case; we can't always split the tasks efficiently) may well outweigh the advantages.

Example

We briefly looked at this example use case model earlier. Here is the context of the model:

> A customer (Applicant) can walk into a bank branch and use one of the e-kiosks to apply for a loan or a mortgage. Applying for some products may need intervention from a Banking Advisor – e.g. if identification papers need to be verified. Customers can also make on-line applications through the

Copyright 2003, Appropriate Process Group

bank's web site, in which case, the Banking Advisor is obviously not available, so an alternative identity verification is required.

The main flow of the use cases is as follows. The name of our system is Application Processor.

UC: Apply for a loan

1. Applicant selects one of the available loan types.
2. Applicant enters personal details. {Capture applicant details}
3. Applicant enters loan reason, loan amount and term.
4. Application Processor calculates the monthly payments.
5. Application Processor displays the loan illustration.
6. Applicant may accept the loan or may change the loan’s details (start from beginning).
7. If the applicant accepts the loan, Application Processor deposits the funds to the Applicant's bank account and passes the loan repayment and bank account details to the Accounting System.

UC: Apply for a mortgage

The use case for mortgage applications is quite similar to the loan application use case but must capture information about the property value and type. There are also more steps and associated business rules for eligibility. We will not elaborate those here.

UC: Capture applicant details

1. Applicant enters essential personal details for identification (name, date of birth, etc.)
2. Application Processor retrieves the applicant's past history with the bank.
3. Application Processor may ask further details depending on the type of customer. {EXT: More Details}
4. Application Processor displays the personal details it has retrieved and asks for confirmation.
5. Applicant amends any details that are incorrect.
6. Applicant accepts the details.

Copyright 2003, Appropriate Process Group

7. If the Applicant provided new information, Application Processor updates the applicant's details.

UC: Capture new customer details

At EXT: More Details

1. Applicant enters personal details (address, etc.).

2. Application Processor verifies the applicant's identification. {Verify applicant’s identification}

UC: Verify applicant’s identification

Application Processor relies on the manual verification of the identification papers.

UC: Verify applicant’s identification on-line

1. Customer selects the type of identification (passport or deferred).

2. For passports, Applicant enters the passport number. Application Processor verifies the passport number and the personal details with the passport bureau.

Alt: Deferred identification

Application Processor displays this message:

Acceptance of the application is subject to verification. You must send three recent utility bills or other official letters to the following address, quoting this number XXXXXX. Original papers will be returned within a working week.

UC: Verify identification at branch

1. Application Processor prompts an available Banking Advisor who goes to the kiosk.

2. Banking Advisor verifies the applicant's identification (manual process).

3. Banking Advisor enters her PIN to confirm her identity, selects the type of customer identification, enters the identification details (serial number, etc.).

Grand Scenario from Chain 5:

Copyright 2003, Appropriate Process Group

Given these use cases, here is the Grand Scenario we would produce for Chain 5.

{Apply for a loan} → {Capture applicant details} → {Capture new customer details} → {Verify applicant's identification on-line}

1. Applicant selects one of the available loan types.

2. Applicant enters personal details. {Capture applicant details}

    2.1. Applicant enters essential personal details for identification (name, date of birth, etc.)

    2.2. Application Processor retrieves the applicant's past history with the bank.

    2.3. Application Processor may ask further details depending on the type of customer. {EXT: More Details} {Capture new customer details}

        2.3.1. Applicant enters personal details (address, etc.).

        2.3.2. Application Processor verifies the applicant's identification. {Verify applicant's identification} {Verify identification on-line}

            2.3.2.1. Customer selects the type of identification (passport or deferred).

            2.3.2.2. For passports, Applicant enters the passport number. Application Processor verifies the passport number and the personal details with the passport bureau.

        2.3.2. Alt: Deferred identification

            Application Processor displays this message:

            Acceptance of the application is subject to verification. You must send three recent utility bills or other official letters to the following address, quoting this number XXXXXX. Original papers will be returned within a working week.

    2.4. Application Processor displays the personal details it has retrieved and asks for confirmation.

    2.5. Applicant amends any details that are incorrect.

    2.6. Applicant accepts the details.

    2.7. If the Applicant provided new information, Application Processor updates the applicant's details.

3. Applicant enters loan reason, loan amount and term.

4. Application Processor calculates the monthly payments.

Copyright 2003, Appropriate Process Group

5. Application Processor displays the loan illustration.

6. Applicant may accept the loan or try another loan type (start from beginning).

7. If the applicant accepts the loan, Application Processor deposits the funds to the Applicant's bank account and passes the loan repayment and bank account details to the Accounting System.

We can form the Grand Scenarios for the other chains in the same way.

The example is obviously quite artificial. I don't expect to see kiosks like these at bank branches for sometime yet and many would not be all that happy about the prospect of some kind of centralised ID verification. The idea of bank staff being prompted to walk around from kiosk to kiosk like checkout supervisors in a supermarket is quite novel, too – even if not particularly practical. On the style side, I kept the alternatives to a minimum, but I would expect a minimal number of alternative paths in this style of use case writing, anyway.

One gap we can see straightaway in these chains is this: What happens to applications with deferred identification? None of the chains tells us how we handle those. *Of course*, we could have spotted this quite quickly in this model, even without the chains. So, is this gap as artificial as the example itself? Not really. I have seen gaps of this sort in many apparently well defined business and system processes. Now imagine having another couple of dozen use cases around these. Would we still be able to spot the gap without forming some kind of end-to-end scenarios with groups of use cases?

A natural reaction to this assertion is that we have too many use cases for such a trivial example. Why not knock them all together into one? If that were the case, imagine the number of alternatives we would have in that one use case, even with this trivial example. Would we be able to spot this gap in a big use case with lots of alternatives?

Or maybe we could have several large use cases that do the process end-to-end, say, "New customer applies for a loan", "New customer applies for a mortgage", "Known customer applies for a loan", "Known customer applies for a mortgage", etc.. Suddenly, these look a lot like the chains that we formed from the model, so they should have the same advantages, too, shouldn't they? How confident are we that we can manage so many use cases with many common parts? Can we keep them all in synch? Easily, reliably and consistently?

Copyright 2003, Appropriate Process Group

This technique gives us the manageability that comes from not documenting the same process segments more than once and still allows us to produce end-to-end walkthroughs of use cases so that we can validate the model without too much effort .

Use case chains, as you can see, provide a convenient way of working with small use cases in large models without losing the occasional benefits of dealing with larger use cases.

Further work on the subject

I don't think the paper fully covers the potential uses for the chaining technique. Here, I mention a couple of tooling issues that need to be resolved not only for this technique to work effectively, but for getting the best out of use case models in general.

Meanwhile, any additions to the subject or corrections are most welcome.

Automation

All the techniques mentioned here are intended for easy automation. The chaining of use cases together is automated now. The rules are simple and involve following relationships from one root use case to another. Automating the generation of the grand scenarios from individual use cases' text is another matter. This is largely achievable in the case of «include» and «extend» relationships. Adopting stricter textual forms in the case of generalisation would make those use cases easy to chain automatically, too.

I have a number of Rose scripts to do some of this work, but they have become rather messy over time. I will publish them on the site as soon as I can get round to tidying them up.

Pre- and Post-conditions

I get asked regularly why we don't have a «precedes» or «follows» relationship in use case models. My answer is that the partitioning strategy of use cases is not based on a "divide and conquer" approach. Instead, the strategy is to keep atomic interactions together and to split use cases only to exploit traits like commonality. Besides, individual use cases should not care about how the system gets to a state over which they have no control; they should care only about whether it is at an appropriate state. If precedence implies that a use case will expect the system to be in some state and it will leave the system in some other state, we already have this dependency captured in the pre- and post-conditions of the use cases, haven't we?

Copyright 2003, Appropriate Process Group

There are significant advantages to being able to chain use cases based on these conditions. This addition would help to highlight and clear up even more gaps. But I wouldn't recommend maintaining the dependencies by hand – apart from turning the model into a fur ball of links, this would risk taking up a significant share of the modellers' time. So, the solution needs to be automation. The difficulty arises because the pre- and post-conditions are usually not described in consistent level of detail. We would need to use consistent naming for both the objects and the states – preferably referring to some domain model. The idea is to look at the pre-conditions of a use case and see if these depend on the post-conditions of another use case.

Some options are:

- *Chaining based on class names:* If use case A mentions class X in its post-conditions and use case B mentions class X in its pre-conditions, B follows A in the chain. We could end up with a multitude of spurious chains in this instance because referring to the same domain object does not necessarily make the two use cases dependent on each other.
- *Chaining based on attribute names:* Same as above, but using attribute names in addition to classes. This would reduce the spurious chains, but we would still have quite a few.
- *Chaining based on state:* If use case A takes the system to a state in its post-conditions and use case B depends on that state in its pre-conditions, we have A followed by B in the chain. This is the level of rigour we need for generating the chains automatically but without producing spurious chains. Given that pre- and post-conditions are rarely ever written with any kind of rigour, this may not be easily automated for sometime yet.

Tool support would help greatly, as the indirect dependencies based on pre- and post-conditions should form at least part of the internal representation of a use case model, even if it is never shown graphically.

Summary

The technique of chaining use cases based on direct relationships and indirect dependencies offers significant advantages over modelling with monolithic use cases. Automatically generated chains can provide the advantages of monolithic use cases (i.e. the end-to-end view captured in a single, accessible artefact) without the disadvantages that come with them (i.e. the inflexibility of working with large use cases and the maintenance issues for common behaviour).

This paper introduced a number of uses for the technique:

Copyright 2003, Appropriate Process Group

- Model validation
- Gap analysis
- Scheduling
- Generating test scenarios

These are areas where chains have already been applied. Other uses will no doubt emerge as more people use the technique.

Current work on this area needs to concentrate on tool support.

Acknowledgements and References

This document is revised as necessary; comments are welcome. Thanks to the initial reviewers: Paul Oldfield.

Thanks also to the following, who provided comments that improved the document:

References

Lieberman (2002)

Lieberman, Benjamin A. (2002). *Putting Use Cases to Work*. Rational Edge, February 2002. http://www.therationaledge.com/

Jacobson (2003)

Jacobson, Ivar (2003). *Use Cases -- Yesterday, Today, and Tomorrow*. Rational Edge, March 2003. http://www.therationaledge.com/

Copyright 2003, Appropriate Process Group