Question Time! about Use Cases

Alistair Cockburn
Humans and Technology
arc@acm.org

Martin Fowler
Independent consultant
Fowler@acm.org

PANEL OVERVIEW

In Britain there is a very successful form of panel session, on a television program called "Question Time." In this programme, four panelists, all public figures, are chosen to give a spectrum of viewpoints. There is always a figure from the two main political parties, usually one from the third largest party, and some complementary figure, perhaps a a business leader, union leader, or media person. This gives a wide spread of views. There are no position statements, the program is filled by discussing questions at about 10 minutes each. Each question is asked by an audience member, all four panelists respond, moderated by the panel moderator (a well known BBC figure). The questioner then makes a brief reply, and there is a brief period of comments from the floor. The questions are submitted in advance by the audience members and the program organizers pick six questions that will be interesting.

This panel on use cases follows roughly the same format. A well known (English!) OO figure will question four expert panelists with differing views, to expose their views and their reasons for differing. There are no position statements, the 90-minute panel is filled by discussing eight questions. The questions are submitted in advance by the public and audience members. The moderator picks eight questions that will be interesting and help tease apart subtle issues and differences between the panelists. For each question, all four panelists respond, moderated by the panel moderator. The questioner makes a brief reply, and there is a brief period of comments from the floor.

PANEL PARTICIPANTS

Martin Fowler, the **Moderator**, is an independent software consultant who has been applying objects to business information systems for over a decade. He has used use cases extensively, but still finds it difficult to crystallize the best ways of defining and using use cases. He is the author of Analysis Patterns and UML Distilled and his clients have included Chrysler, Citibank, and Netscape.

Ivar Jacobson, is the inventor of use cases, founder of Objective Systems and the Objectory method, and one of the "three amigos" of UML. Other than the UML definitions and numerous articles on use cases, he is well known for his books Object Oriented Software Engineering, The Object Advantage, and Software Reuse - Architecture, Process and Organization for Business Success.

Bruce Anderson, formerly an academic and currently senior consultant in the IBM Consulting Group, was the originator of the architecture handbook project. He has been investigating use cases for his consulting work and has no particular vested interest in any one definition.

Alistair Cockburn, consulting fellow at Humans and Technology and special advisor for the Central Bank of Norway. His "Structuring Use Cases with Goals" has been used as a basis for introducing use cases in many companies. He used this use case model to coordinate development of a successful, 200-use-case, 60-work-year project. He recently published Surviving Object-Oriented Projects.

Ian Graham, Vice President, Chase Manhattan Bank, is the inventor of the SOMA method and 'task scripts', which are similar in spirit to use cases. His books include the SOMA book and The OPEN Process Specification.

I heard last year the sentence, "The methodology wars are over; let the use case wars begin." This showed the high level of interest, discussion and disagreement on the issue. If the topic of use case is suitable, I don't think the usual panel format is. The opportunity here is to let the audience see how the use case pundits differ, and to preselect the questions to get ones that will be of general interest.

POSITIONS

Martin Fowler

As moderator, here are some plausible questions I see.

- *How big is a use case?* Recently I worked with a project that is developing a payroll system. The project was roughly a dozen man-years of effort. They used around 170 use-cases to describe what they were going to do. Another authority on use-cases was surprised by

that number: he would have expected 25.

- *What makes a good use case?* I'm not sure what the areas of discussion are here, but it is an important question.
- *How much detail do you do up-front on use cases?* The question is how much detail do you need in the use cases before you start incremental development of those use cases. I have seen variations between a couple of paragraphs on a card, to a document section of 10-20 pages.
- *Do use cases lead to functional designs?* A number of people have seen use cases encourage a functional decomposition design. Do you find this a common problem? How do you avoid it?
- *Do you have internal use cases?* For some people, a use case must be something that involves user interaction. For others it can be a piece of internal functionality (such as a database interaction use case).
- *What do you use use-cases for?* Requirements gathering, project planning (assigning use cases to increments), system testing. Are they good for all of these? Anything else?
- *Do you use actors?* Jacobson's approach ties use-cases to actors. Do you find actors useful?
- *Do you use extends and uses relationships?* Jacobson's approach makes use of extends and uses relationships between use cases. Do you find these useful, do you find them dangerous?
- *How formal should use cases be?* Some people break down uses cases into sub uses cases and come up with defined steps. others just write a few sentences to explain the basic idea.

Ivar Jacobson

Use cases were first made public in the OOPSLA'87 paper: Object-Oriented Development in an Industrial Environment, but they had by then been used in a similar form for many years. Definitions were given for practical uses for real products. Later in summer 1992 a full description of the use case construct and use case driven development was published in the Object-Oriented Software Engineering book. More than 200 pages described use cases and their realizations as collaborations of a society of objects, playing roles in realizing the use cases. Use case driven design was introduced, meaning that the use cases drive the whole development work. Every use case found in requirements capture is mapped to a realization of that use case in analysis or design and finally to a set of test cases in test. The ability to drive development and not just help in requirements capture is the great value of use cases.

When writing the book, I was very careful in not trying to formalize the use case idea. Working with formal specifications for 15 years by that time, I had seen what too early formalization can do to good ideas. To formalize is both very simple (you can do it as a master thesis at some good computer science college) and very difficult (what could be formalized and still practical). In the book I made it very clear that use cases were classes, that could be instantiated and that could interact with actors (users) only. Interactions between use cases within the same system being modeled were prohibited for strong reasons. Use case instances were atomic so that conflicts between different instances were avoided (they could not be modeled here but elsewhere).

Use cases could be related through two kinds of class-relations: uses which is a generalization-specialization relationship and extends which is a mechanism by which you can extend behavior specified in another use case. Both these relationships are very important to avoid redundancy and to allow evolution.

Use cases could be described by using interaction diagrams with actors and the use case itself as participants. And, a use case could be described by activity diagrams (we called them state transition diagrams, but I use the UML term here) or by a state machine (similar to state charts). These techniques were further elaborated in a series of articles in ROAD in 1994-95.

Use case realizations were described by interaction diagrams with objects of specific classes being participants, playing roles in the realization. This technique had been used since 1967 to describe collaborations among objects, so it was very well proven.

Thus use cases and their realizations were as well described as anything else in object-oriented design. The major reason why people still think they need to be further formalized is that they are not relying on the semantics of a programming like Smalltalk, C++, Java – at least not yet!

In the work on UML we have made a precise definition of the use case related language constructs. Use cases are classes with attributes and operations, participate in collaborations with actors, are realized as collaborations of classes or subsystems – each playing a role in realizing that use case, have class relationships, can be described by a state chart or an interaction diagram.

The idea of use cases has moved over to the world of business engineering. A business use case is a true business process. From a business model of an organization we can derive the system use cases required

to give IT support for the business.

Use cases are also very important to achieve substantial reuse when building a suite of applications. Use cases are reusables. They help in building systems of systems with traceability between higher level systems and lower level subsystems. Use cases also serves as the problem specification part of patterns, which are the solution part.

However, most important is the practical acceptance and application of use cases. With or without formal definitions, users have picked up the basic ideas and intuitively applied them. There are 100's of reports on the successful use of use cases in commercial projects, many of them not based on any particular formalism. This is very essential and most encouraging for the future of the ideas.

Bruce Anderson

I take a very pragmatic view of use cases, so here are some things that I've found

- Remember what use-cases are for. They are something that users and developers create together to help them agree. There is always an element of "you know what I mean". Sometimes some of this needs spelling out, as when we did them for a website and it was clear that you could bookmark yourself at any point in a course and come back later, perhaps many times.
- A requirements document should do the job, and needn't be just use-cases. For a call-centre, include a model of call states - it would be crazy to "code this up" as use-cases.
- The context for use-cases includes the business you're in, the team you have and the kind of systems you build. Develop your own style that works for you.
- I've found Alistair Cockburn's form very useful. It makes sense, and it's good to have his paper backing it up.
- The name is a hindrance for business people. Sometimes I say that it's an anglicisation from Welsh or Japanese, to emphasise that "use case" is not an English phrase.
- You can make use cases only for a system that you can imagine already. If you are at the stage where you (the end-user) are wondering what kind of system will fit the bill, then you need to do some envisioning. This is a quite different thing. Correspondingly you will have to give your use case readers some context so that they can do the imagining. Make sure you have a preamble that does this.

Alistair Cockburn

Reponding to Martin's suggested questions and other natural once, I would answer:

A use case is a collection of scenarios, bound together by a common goal. Scenario is a trace of actions and messages from trigger to goal completion or abandonment, non-branching but possibly with parallelism. Actor is actually a role, the thing or person having the goal. There can be one use case for every system trigger.

Use cases are for functional requirements of any system. They should be written black-box style. Collaboration diagrams show the white-box workings, but that involves the design, not just requirements, and therefore come later in the development process. Use cases can be used to gather functional requirements of companies, departments, computer systems - any system.

A non-negotiable value to me of use cases is that they are easily read by any normal person, the target audience is the person expert in the use of the system and the person designing the system. They should be fairly easy for a non-technical person to write. For this I am willing to give up formality and precision in the writing and tolerate ambiguity. Use cases have a formal structure, which one is free to abuse in the name of readability.

How big is a use case? As big or little as it needs to be. There are giant use cases and teeny use cases. I use one per business task, typically also a system transaction. I put them into several levels, depending on what system is being described. Those describing the corporation as system are not detailed enough to develop the software from. Those describing the UI are too detailed to serve as overall, readable, early-written functional requirements. Those below business transaction level are too detailed for readability and should only be included as they are shared by other use cases.

What make a good use case? That the goal, the value of satisfying the use case, is evident. That the use case is written with steps at the same level, making it easy to follow. That the trigger and preconditions are clearly evident. That it is relatively brief (1-2 pages). That the set of use cases clearly cover the function space.

How much detail up-front? Unclear question. Before writing the first one or before signing off on them or before designing? All different questions. Start writing right away, first thing, immediately. Start designing as soon as the set for the initial release are identified. Start analysis only after at least one use case is sketched. Continue to evolve each use case over the life of the design.

Do they lead to functional designs? Not necessarily.

Are there internal use cases? Can be. If you scope a subsystem as The System Under Design, you can bother to write use cases for them. Most people don't find it worth

the effort, but there is nothing contradictory in it and I have seen it done.

Do you use actors? I work in three stages. At the first stage, actors are very important. They help define the breadth of the system, showing whose goals need to be accommodated. In the second stage, they are completely unimportant; the goals drive the work; actors start to overlap, become generic and meaningless. At the third stage, there is a mapping from real people or job descriptions to goals, and these real people become actors again, in a different and weaker sense than at the beginning. But the mapping from job descriptions to use cases is important.

Do you use extends and uses? Yes

How formal? Formal substructure which can be abused in favor of readability, writing done in plain prose, not pseudo programming.

Ian Graham

The lack of a precise and universally accepted definition has led to a proliferation of approaches all calling themselves 'use case' based. I once asked if a precisely defined notion of use cases (such as task scripts) should use a different name. The consensus that emerged was, yes, we should really rename the idea but, no, the industry had bought the idea of use cases and so we had to live with the label – however ill defined. I disagree with this pessimism and continue to talk about task scripts, while recognizing our deep debt to Jacobson.

A relatively simple application generated hundreds of use cases; far too many to be manageable! One of several reasons for the tendency to find too many use cases is the lack of any notion of essentiality or genericity. This defect is potentially very serious. A task script represents a generic use case. The same application had only ten task scripts: an order of magnitude improvement!

In a system that prices orders and provides quotations or diverts them to a salesman for manual pricing or quotation, it would be normal to have four use cases at the highest level of abstraction: autoprice quote, autoprice order, manual order and manual quote. This is not parsimonious because the applying diversion rules that decide whether to re-route orders or quotes to the sales desk may be different for quotes and orders while the notification process is the same (or vice versa). It is better to have independent task scripts for the different components of the use case, which can then be assembled to model complete business processes. This promotes greater re-usability of task objects. SOMA sequence diagrams show the way *user tasks* are composed into end-to-end processes, while UML sequence diagrams focus on the operations of classes and are therefore intrinsically unsuitable for this purpose.

Severe problems arise if we try to treat use cases as objects. The structural links between use cases, uses and extends arrows, point in the wrong direction from the point of view of encapsulation. Therefore we are not able to treat use cases as bona fide objects. Therefore they are not really reusable. Usage links give us a much better way of handling exceptions than is available with standard use cases, which suffer from the poor semantics of extends.

As project managers we would like to believe that the number of use cases in a requirements analysis gives some indication of the business benefit or the final system or even of the amount of effort involved in building it. However, use cases are notoriously hard to measure, since a use case can be any length. The lack of a notion of atomicity means that no metrics can be reasonably defined for use case models – unless we change their definition as many companies have indeed done: numbering the sentences of a use case for example. The task script notion includes a notion of atomicity that permits developers to measure task complexity by simply counting the atomic scripts.

Use cases may be presented without business goals. This hampers traceability efforts and impedes the understanding of users. I would add that the absence of the notion of triggering events is also a deficiency. Task objects have their scripts triggered by events in the agent model and that messages in that model have explicit goals. I think that this is a better solution than attaching the goals directly to use cases as suggested by Cockburn and by Rawsthorne inter alia. Clearly the goal or contract is in the mind of some agent rather than being something possessed by the task itself.

Use cases have been criticised for only dealing with the external, interface aspects of systems. This is deliberate. The intention is to focus designers' attention on what the system did for its users, rather than how it did it. However, a variant on use cases that permits internal modelling can be desirable in some circumstances.

Finally, there is even a problem with the phrase use case itself. It makes for very unwieldy and unprosodic expressions. What is needed is an approach with a sound, theoretical basis and a precise definition. We need a notion of genericity and a notion of atomicity for use cases or their equivalent. Use cases have no clear link to a business process model and are offered as such a model in their own right, which I feel does not make good sense. Finally, we need an approach that is properly object-oriented in supporting encapsulation and inheritance.