FAQ: Modelling

Topics:

- *When do we stop modelling? We could go on and on ...*
- *What's a Use Case?*
- *What's the best way to write Use Cases? The many books out there each seem to push a different version*
- *What's the best tool for drawing Use Cases?.*
- *How can we package use cases?*
- *What is UML anyway?*
- *What's the relationship between UP and CMM (i.e. the Unified Process, and the Software Capability Maturity Model)?*
- *All this modelling stuff and process stuff is just pie-in-the-sky nonsense. It is hot air, useless, wasteful, and expensive to boot! We don't bother with any of it.*

When do we stop modelling? We could go on and on ...

Stop when your model holds up to the difficult "what if" questions that you, your colleagues, and clients throw at it. This will take several iterations but will show your model is robust and therefore stable. It'll handle the additional requirements you're all able to imagine. That's good.

Stop when you are reaching agreement with your colleagues. Other software architects look at your model and try to understand it. And try to tear it apart. Your model holds up to this stress testing. That's good.

Stop when you are done translating user requirements. And conflicts have been mostly resolved. Stop when you're getting consensus from clients, and users. That's excellent.

A model, by definition, will never be perfect. Someone once said “All models are wrong! Some are useful.” You'll return to a previous model and refine it over time. But as long as the above conditions hold, you can move on to the next stage of the development cycle.

What's a Use Case?

A use case is essentially an "instance of use". It describes the sequence of steps a user follows in getting a certain task done. So the implementation of all the use case you've defined, is the software you deliver to your client.

Use cases can be seen as requirements-in-context, of which more below. A use case model consists of a drawing that illustrates which users interact with which use cases. The users are known in the jargon as Actors and may include non-human things such as third-party software your system must interface with, and any sensors, actuators, and timers.

But a use case model is more, much more, than a drawing. Each constituent use case must be accompanied with a textual description. The use case description is a sequence of actions performed by the user and the system, that yields an observable result *of value to the invoking actor*. This is the most important part of use case modelling, and it is here that most of your effort should be devoted. See the section immediately below for advice on writing a use case description.

Use cases are part of the UML - the Unified Modelling Language. See below.

What's the best way to write Use Cases? The many books out there each seem to push a different version.

Ah that old chestnut... What's best? ... It all depends... And I'm not sitting on the fence.

It depends on whether you and your colleagues involved in analysis and design have written use cases before. What's worked for you in the past? And what about your main contacts on the client/customer/user side? Have they seen use cases before? What's worked for them? Identify a common basis and develop a template. Run it past your colleagues and client. Improve it. And then use it.

Initially write a short paragraph that tells a story about how a user would typically go about doing a task with the system. Don't get into the details; ignore the special exceptional events and error conditions, for now. In time, elaborate on this initial description. As the people involved become familiar with use cases improve your template. Add the following:

- name of event that triggers the use case and causes it to be executed
- Primary and Secondary Actors involved with the use case
- Pre-conditions that must hold before the use case can run
- Alternatives and Exceptions to the normal flow of events
- Supplementary i.e. Non-functional Requirements (such as performance, safety, security, usability, etc.)
- Post-conditions that shall hold true after the use case finishes execution.

As Use cases are requirements, consider adding meta data, such as:

- Use Case Number (a unique identifier)
- Version number
- Author of the Use case

- Source of the use case (parent document, or a person)
- Frequency of use (a measure of how critical the use case is).

Such a comprehensive description of a use case can be seen as a requirement in context. The latter is provided by the descriptions of the actors involved, and supplementary requirements.

What's the best tool to draw Use Cases?

Initially you will want to draw and re-draw your model frequently. The best tools for this are a pencil and an eraser. You'll probably want to change the names of use cases and actors, add and remove use cases, and change their inter-relationships (associations). Easily done with pencil and eraser. Or use a flip-chart (easel) , Post-Its and pencil. These are great for working with other stakeholders – they can easily see the evolving model, even from a distance, without peering over your shoulder! And you still have the flexibility of re-ordering your Use Case model by simply moving the Post-Its. Finally you can draw in the arrows associating the use cases and actors.

When the model is ready for wider review, you will want to draw it in a software tool. If you're working on a new project, there are unlikely to be other analytical models with which to integrate. Use a drawing tool such as Microsoft Visio.

When you've incorporated any feedback into your model, and you're ready to move on to producing more models such as Object, Class, and Interaction models, consider using a specialist modelling tool, such as Together, Rational Rose or Enterprise Architect. Your models can now be enhanced and inter-linked and you can use the tool's code generator to produce code stubs. And as the complexity of your models grow, you can keep them up to date.

How can we package use cases?

That depends… Are you producing libraries? Then begin by isolating those use cases which are independent of any application and put these into a package.

Do your use cases have a natural grouping? Can they be grouped by actor, or by type of functionality? Group these together. Putting those that provide back-office functionality together, is one approach.

Do you have a distributed system? Then you can separate them by node such as client and server, or by tier.

Think about cohesion and coupling. Those that have strong relationships and dependencies, must be grouped together. You must try to develop packages that have high intra-relationships but low inter-relationships.

What is UML anyway?

The Unified Modelling Language is a set of 13 graphical notations. These notations enable one to be very precise in describing different aspects of our system. This rigour allows one to draw a specification rather than write one.
The diagrams are: use case, class, object, sequence, communication, state, activity, interaction overview, timing, component, deployment, package and composite structure.
UML is not a programming language.
UML is not a software method.
UML is not exclusively for Object Oriented development.

To an extent it's old wine in a new bottle. Notations like use cases and states, can be used wherever. And the sequence, activity, component, and deployment diagrams can be easily adapted for use wherever.

What's the relationship between UP and CMM (i.e. the Unified Process, and the Software Capability Maturity Model)?

In a word: None.

The Unified Process is a step by step approach to developing software. It prescribes the start point, the modelling notation (UML - the Unified Modelling Language), the intermediate deliverables, the specification documents, the analysis and design and testing artefacts. It helps you develop complex software in an iterative manner. It's a use case driven approach. It's what those who get their fingers dirty in actual software development need to know.

The Capability Maturity Model is a software process improvement (SPI) framework. It advises on how you can bootstrap your development group to deliver software to time, budget, and quality goals. The CMM advises that the first step in any SPI plan must be to define and follow managerial processes. Only after a subsequent assessment shows your group has does this effectively, should you define and follow technical software engineering processes.

So you see, UP and CMM are quite different.

All this modelling stuff and process stuff is just pie-in-the-sky nonsense. It is hot air, useless, wasteful, and expensive to boot! We don't bother with any of it.

And you're making money? Kool. If you tell the world the secret of your success, you'll be making even more money. Believe me. Hey, as some bright guy said: if it's not broken, don't fix it.

I suspect however, this is a case of the pot calling the kettle black.

For us mere mortals, SPI, UP, UML and such approaches are not an option. Let's consider the objections in turn.

It's pie-in-the nonsense. I don't agree. SPI, UML, UP and such approaches have been developed by contributions from thousands of practitioners worldwide. They're based on experience. Lessons have been learnt from the many mistakes made on countless software projects. The net outcome is embodied in such approaches.

It's hot air. I don't agree, This stuff is well thought out. It's documented. It's internally consistent. It's very detailed. It has predictive power.

It's useless. No sir. Quite the opposite. It is useful, very useful. It's practical. The notations are invaluable in clarifying one's own thoughts. And very useful in establishing a common understanding amongst the development team. And in developing a shared understanding with the client. And in giving the project leader fine-grained tracking tools, to track the course of development, control deviations, and manage a successful team, and a successful relationship with the client.

It is wasteful. Hmmmmm, you may have a point. When adopted verbatim, yes, they can be wasteful of time. The key is to Adopt and Adapt. Don't leap in with both feet. First establish your success criteria. These must be measurable. Then take a nibble. As some one said: Don't bite more than you can chew! For starters, try a bit of UML. Develop some use case models. Use paper, pencil, and an eraser. Or use a whiteboard. An electronic whiteboard will be expensive, but better still. Write some use case descriptions. Adapt the approach prescribed in text books, based on what works for you. Work with a sparring partner. Stress-test your model. Refine it. This will reap dividends in a better model. It won't be wasteful.

It is expensive. True - if you buy a software tool. These are seriously expensive. Not in the initial outlay of a few hundred pounds, but in the total cost of ownership. Use a software tool too early and you'll end up developing wonderfully formatted models, which are meaningless and incorrect. And that is seriously expensive. Economists have a useful concept. They calculate the cost of *not* doing something - the Opportunity Cost. Is the cost of developer time spent on modelling during software development, greater than product re-design costs incurred after delivery? Can you calculate the effect of faulty software on the relationship with your customer, and on lost business? Chances are you're spending loads already. Why shut the stable door after the horse has bolted?

For the latest version of this FAQ, please visit:
http://www.ash-consulting.com

