Requirements capture and use cases

Perdita Stevens

School of Informatics
University of Edinburgh

Requirements capture

You've already seen some small examples of the difficulty:

- not being sure what was required in programming exercises
- as a user, software not doing The Right Thing...

In small systems, fairly easy. Not sure? Go and ask.

In large systems, much harder – need to budget (time and money), negotiate scope; difficulty of resolving questions; significant effort needed before anything can be demonstrated and checked.

Inadequate requirements capture is the major source of project failure (e.g. according to Standish CHAOS reports).

Formal and informal approaches

There's a wide spectrum, including for example:

- just intuit (OK if you're writing for yourself)
- user stories (Extreme Programming)
- use cases, with some level of natural language documentation
- formal specification

Today we focus mostly on use cases, but before that...

User Stories

Used in "agile" (low ceremony, lightweight) development processes e.g. Extreme Programming (XP) – more on process later.

Brief, written by customer on index card. E.g.

`10. User A leaves the office for a short time (vacation etc.) and assigns his access privileges to user B, so B can take care of A's tasks while A is gone.` *Source: user, Risk, M*

Pros and cons of user stories

Pros:

- can really be owned by the customer: so more likely to be correct
- quick to write and change
- small, so relatively easy to estimate and prioritise

Cons:

- May be incomplete, inconsistent
- Only work in conjunction with good access to the customer
- Not suitable to form the basis of a contract

Now we go on to medium-ceremony approaches.

Requirements for what?

Traditional to distinguish *functional* from *non-functional* requirements.

Functional requirements: What the system should do.

Non-functional requirements: How fast it should do it; how seldom it should fail; what standards it should conform to; etc.

Note that non-functional requirements may be more important than functional requirements! (Workarounds...)

User and system requirements

Two ends of the requirements spectrum:

User requirements: What the user will require of the system, or how the customer requires the system to behave to the user. Often written in natural language, perhaps in some structured form.

System requirements: Specifies the system behaviour in detail, for the implementors. May be basis of contract. Often in semi-formal language such as UML.

Use cases

document the behaviour of the system *from the users' points of view*. They help with three of the most difficult aspects of development:

- capturing requirements
- planning iterations of development which are good for users
- meaningful system testing

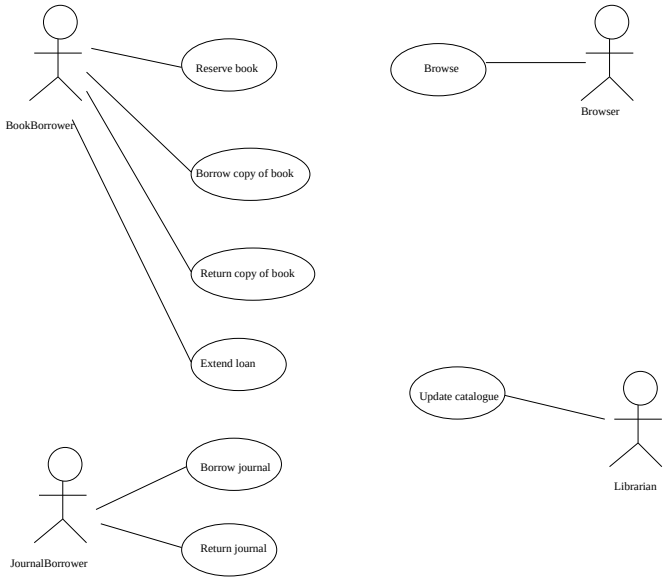
First introduced by Ivar Jacobson (early 90s), developing from *scenarios*. Independent of OO – strength or weakness??

Simple use case diagrams are easy to understand: can be useful for communication between customers and developers.

Whose requirements?

- Who is it that requires things? Directly or by proxy?
- Who else might be affected?
- Who is it that approves requirements?
- Who do you write requirements documents for?
- Who pays?
- Who can cancel the project at each stage?

A simple use case diagram



Actors

An **actor** – shown as a stick figure – can be:

- a human user of the system *in a particular rôle*
- an external system, which *in some rôle* interacts with the system.

More specifically, a particular *kind* of user. E.g., bank has many customers, but we only show one Customer actor on the diagram.

The same human user or external system may interact with the system in more than one rôle: he/she/it will be (partly) represented by more than one actor. (e.g., a bank teller may happen also to be a customer of the bank).

Requirements capture

Use cases can help with requirements capture by providing a structured way to go about it:

1. identify the actors
2. for each actor, find out
   - what they need from the system
   - any other interactions they expect to have with the system
   - which use cases have what priority for them

There may be aspects of system behaviour that don't easily show up as use cases for actors.

What is a use case?

A **task** involving the system which has value for an actor, e.g. `Borrow copy of book`.

Shown on diagram as named oval.

Also includes (textual) description of the (a?) sequence of messages exchanged between the system and any actors, and actions performed by the system, in order to realise the functionality.

Connection between use case descriptions and other forms of requirements documentation is rather controversial.

Use cases: scope and connections

A use case:

- may include logic to handle unusual or alternative courses, e.g. "if the `BookBorrower` has the maximum number of books on loan already, refuse this loan" *even though these may result in the actor being unsatisfied*;
- may be associated with other UML models which show how it is realised;
- includes text which may reference other requirements documentation.

A use case diagram summarises all the tasks performed by the system (or subsystem, etc.)

MCQ (from 05/06 exam)

In a use case diagram for a system, an *actor* may be:

1. a user of that system
2. an object in that system
3. either of the above
4. none of the above

Politics

If we capture requirements in terms of use cases, we should understand *what is important to whom*.

Make sure system delivers added value:

- soon
- to all the people who might scupper it
- in every iteration

Result: the project isn't cancelled. Supposedly...

Analysis vs design

Some actors are part of the requirements: usually the ones who derive benefit from a use case.

Others are part of the (business process) design: the ones who interact with the computer system to provide the benefit.

For example, consider a `FindBook` use case of a library, in which the user enters details of a book and wants to end up with a copy of it. Maybe the system will give the user directions to where the book is on the shelf. Maybe it will alert a librarian to go and fetch it. In the latter case, should the librarian be shown as actor? In some sense, the choice is a design decision.

What use cases are not

Use cases document the requirements of a system: not the whole business process into which the system fits.

For example, UML does not permit associations between actors: you cannot legally use a use case diagram to show an interaction between two humans followed by one of them using a system. (E.g. can't legally show librarian and library member as separate actors in Borrow Book, if only the librarian interacts directly with the system.)

There are extensions to UML to allow business process modelling, not considered here.

Using use cases in development

Use cases are a good source of system tests: requirements documented as desired interactions, which translate easily into tests.

Earlier, they can help to validate a design. You can walk through how a design realises a use case, checking that the set of classes provides the needed functionality and that the interactions are as expected.

Use cases are not limited to documenting the whole system: they may describe any classifier, e.g. subsystem, class, component.

Reading

Required: SWEBOK 2004, Chapter 2, Software Requirements.

Suggested: Somerville Chapters 6 and 7

Suggested: Stevens Chapter 7.

Suggested: Start browsing `http://c2.com` starting with link on course web page.