From requirements to modelling 1: Conceptual modelling

Perdita Stevens

School of Informatics
University of Edinburgh

Plan

- What's a conceptual class model?
- When and why do conceptual class modelling?

What is a conceptual class model?

Aka domain model – some authors mean slightly different things by these two terms, but they are essentially the same thing.

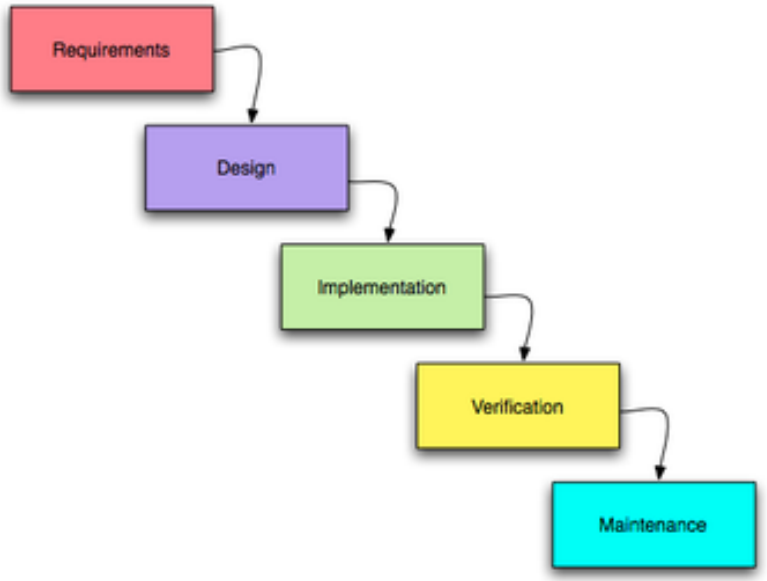
A model that records the key domain concepts and their relationships in the domain.

Or: the main *things* your users talk about, and how they know they are connected.

Does not record things that reflect only *this* system's requirements ⇒ robust to changing requirements.

Reference for the vocabulary you'll use.

Remember the waterfall model?



Conceptual modelling in the waterfall

Nobody truly uses a waterfall process, because you can't completely settle the requirements before doing anything else.

However, it's a useful strawperson process for understanding the contribution of activities.

"Software analysis" now old-fashioned term – subsumed partly under Requirements and partly under Design – but still a useful idea.

This is where conceptual class modelling fits.

Make sense of the world in which the requirements fit, in order to design a system.

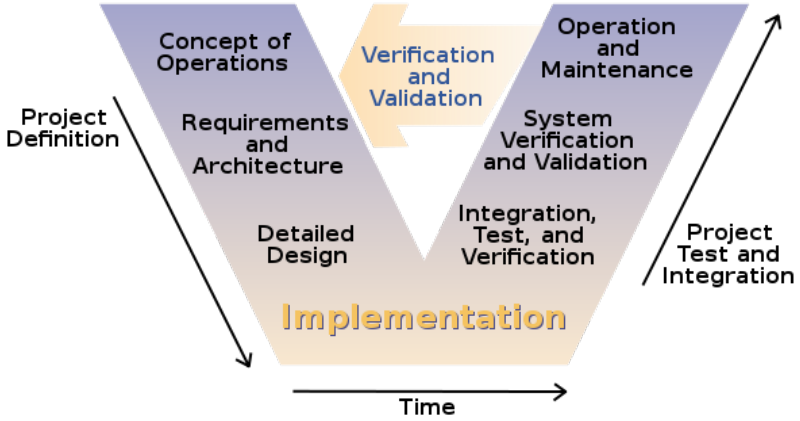
Process in a nutshell

This is not a course about development process! Brief excursion on what you need to know...

- waterfall never worked;
- real processes are iterative;
- they vary in how the iterativeness is controlled;
- high ceremony, e.g. V model, Rational Unified Process
- low ceremony, e.g. most agile processes, e.g. Extreme Programming

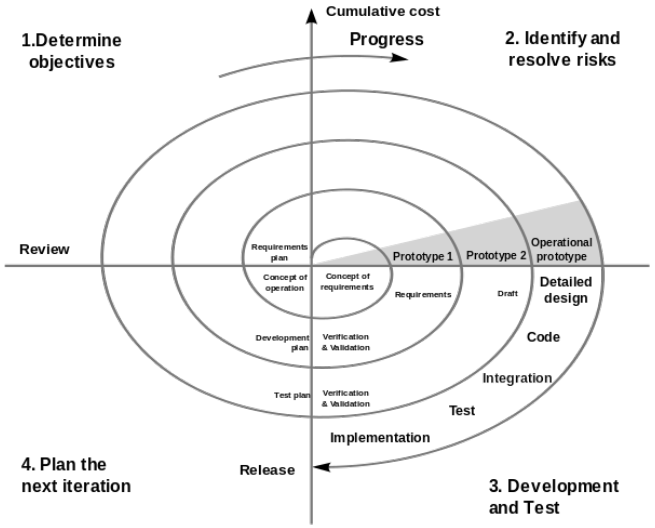
The future: combine agility with modelling...

V model



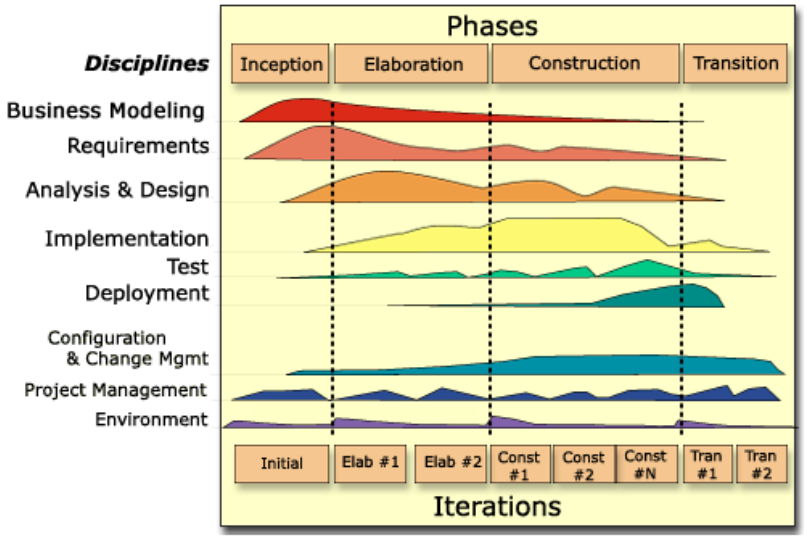
https://commons.wikimedia.org/w/index.php?curid=10275054

Spiral model



https://commons.wikimedia.org/w/index.php?curid=9000950

Rational Unified Process



Modelling in the SE process

In this course we are largely process-agnostic

but we assume that models are first-class artefacts

supporting quality in design

That may mean

- key diagrams live on the whiteboard and get updated;
- or they're signed off in a design document;
- or they live in a tool and code is generated from them.

Artefacts to end up with (eventually)

1. Complete set of use case descriptions, summarised in a use case diagram.
   Each use case description describes, step by step, the required interaction between the actors and the system.
   It describes both the usual ("sunny day") scenarios, and any alternative scenarios (e.g., what should happen when things go wrong).
2. A conceptual class model that forms the basis of the system design.
   The classes in the model must have appropriate attributes, associations and operations (this is the hard part!)

Which comes first?

The use cases, or the conceptual class model?

Really both:

- need some idea of requirements, i.e. actors and use case names, to get started;
- key domain concepts emerge as you learn details of use cases;
- it's *very* helpful to keep the terminology of the use case descriptions and the conceptual class model consistent;
- so refine them together, until both are solid and consistent.

Reminder: noun identification

In Inf2C-SE you met the idea of identifying candidate classes by underlining noun phrases in a system description, then eliminating things that weren't classes.

This is still the key idea. We add identification of

- relationships between classes and objects (associations, generalisations)
- data associated with objects (attributes)
- constraints on configurations of objects and their states
- and later, behaviour of objects (operations)

Suggested follow-up

Read up on the processes mentioned (Wikipedia articles are good starting points).