Modelling and design

Perdita Stevens

School of Informatics
University of Edinburgh

Plan

- What is meant by *modelling* in software design, and in SE more generally?
- Why is modelling important?
- History of modelling
- Diverse uses of modelling in current software engineering practice

Note on spelling

modelling: British English

modeling: American English

Complication: UML stands for the Unified Modeling Language, which is a trademark, and so is spelled the American way even in the UK.

What is a model?

A model is an abstract, usually graphical, representation of some aspect of a system.

(Example paper coming up: guess the date?)

Iterative multi-level modelling - a methodology for computer system design

Abstract. The paper presents a method of modelling a computer system design as it evolves, so that evaluation can be made an integral part of the design process. The paper introduces the concept of concurrent existence, within a single model, of several representations of the system being modelled, at differing levels of abstraction. Thus important design decisions are expressed directly in terms of appropriately abstract quantities, facilitating understanding, validation, and modification of the system design. The paper includes brief details of an experimental implementation of the modelling technique and of the use of the technique to model both hardware and software components of a multi-processing system.

The reference

Zurcher, F.W. and Randell, B. Iterative multi-level modelling, A methodology for computer system design.

IFIP Congress, Edinburgh, August 5-10, [date redacted]

Graphical modelling

In some sense older than SE.

Organisation charts, PERT/CPM diagrams, genealogy diagrams used as analogies in early papers Earliest real SE diagrams are data-oriented.

Bachman 1969 The [DSD] has been used fruitfully over a period of five years by a limited but rapidly growing audience

Why model?

Most people find it psychologically "natural" to visualise software somehow. Can help:

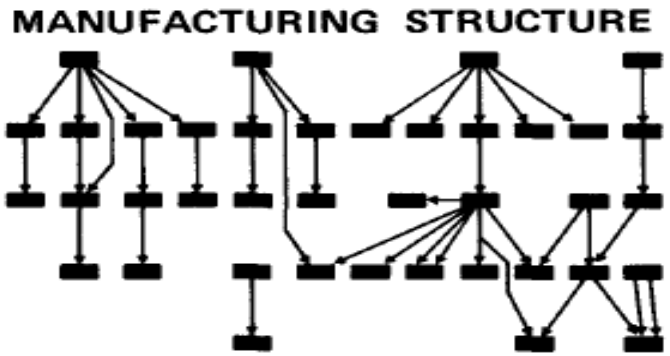
- the person building the software to keep it straight in their head
- communication between groups of people (developers, customers...)

Tensions:

- between flexibility and standardisation
- between having obvious meaning and being precisely expressive

Let's look briefly at the history of modelling software.

A "very large" model, Bachman 1969



Use of models

- Static (structure) models: early on, seen as pictures, not as formal objects. (Chen, 1976 emphasises the (novel?) idea that a diagrammatic representation could be isomorphic to a symbolic one.)
- Dynamic models. E.g. flowcharts (e.g. Gilbreth, 1921); automata (e.g. Taylor Booth's 1967 book Sequential Machines and Automata Theory); Harel statecharts, 1980s; message sequence charts, 1990s.

Thus modelling software was always around.

But really took off with, and became identified with, OO.

Object oriented modelling

1980s/early 90s: explosion of OO

Plethora of gurus, each with own company, tool, book, modelling notation

including Booch, Rumbaugh, Jacobson, the "three amigos" who originated UML, the Unified Modeling Language

and Coad, Odell, Schlaer and Mellor, Wirfs-Brock...

Unified Modeling Language

"The methods war is over: we won" said Booch and Rumbaugh.

OMG pulled the community together into standardising UML 1997 UML1.0... 2010 UML2.3...

UML now completely mainstream

Unified Modeling Language

Thus the flexibility/standardisation tension has (for now, in the mainstream) been resolved in the standardisation direction.

(But there are mechanisms for customising UML, and related ways to define *domain-specific modelling languages*: may come back to this at end of course if time.)

Good:

- Thriving tools market
- Mobility (of people and projects) - sort of
- Critical mass enables development of techniques and new kinds of tools to work on UML

Bad:

- You all have to learn UML :-)
- Bloat
- Inertia

Where is the pain in software engineering?

Hint: it's *not* felt during design.

Requirements ("cause" of most problems with software)

Maintenance (90% of cost of software ownership)

Why do we waste time teaching modelling and design (and testing) then?

Requirements

Eliciting requirements for software is very, very hard.

- Customers genuinely don't know what they want.
- You may not have customers yet.
- You may not get access to the right people, or ask the right questions.
- Requirements change all the time anyway!

There are things you can do! But you will still get it wrong.

Good modelling, design and testing should let you change the software quickly and without breaking it, when the known requirements change.

Maintenance

Evolving software over a long time is costly and often hard.

- Software rots - it is based on assumptions that become false (that supporting software e.g. components and compilers continue to exist, be supported, work, work together; that requirements are as captured; that old architecture will support current conditions).
- People move on (someone understood this - but they're not here now).

There are things you can do! But it will still be costly.

Good modelling, design and testing should let you change the software quickly and without breaking it, when things change.

(Major challenge here: not "losing" the models during the development. More on this later...)

Styles of modelling

UML use varies across projects and organisations, e.g.

- people scrawl UML diagrams on napkins and whiteboards
- UML diagrams appear in documents (sometimes after the code has been written)
- UML diagrams are developed in tools before the code, and code is generated from/in parallel with them

Not quite the same as Martin Fowler's classification (required reading)

- UMLAsSketch
- UMLAsBlueprint
- UMLAsProgrammingLanguage

To Do

1. Schedule time into your week for SEOC work. You need *at least* two hours per week. ("Private Study/Other 78")
2. Read through this week's lecture slides. Ask (your colleagues, failing that, me) about anything you're unsure of.
3. Read Martin Fowler's page `http://martinfowler.com/bliki/UmlMode.html` and the pages on the three ways of using UML.
4. Revise the Inf2C-SE material on requirements, including use cases.
5. If in doubt: watch the use cases video.
6. Watch the class diagrams and sequence diagrams videos.