Software design and modelling

Perdita Stevens

School of Informatics
University of Edinburgh

What is design?

For example: what are the classes in your system?

Design is the process of deciding how software will meet requirements.

Usually excludes detailed coding level.

What is good design?

A quotation from Donald Schön

> Designers put things together and bring new things into being, dealing in the process with many variables and constraints, some initially known and some discovered through designing. Almost always, designers' moves have consequences other than those intended for them. Designers juggle variables, reconcile conflicting values, and maneuver around constraints – a process in which, although some design products may be superior to others, there are no unique right answers.

Donald A. Schön
Educating the Reflective Practitioner
Jossey-Bass, San Francisco, 1987.

Which of these two designs is better?

1.

```
public class AddressBook {
  private LinkedList<Address> theAddresses;
  public void add (Address a) {theAddresses.add(a);}
  // ... etc. ...
}
```

2.

```
public class AddressBook extends LinkedList<Address> {
  // no need to write an add method, we inherit it
}
```

Design principles 1

If you had to pick two, they'd be:

- maximize coherence
- minimize coupling

(Why?)

Design principles 2 (from GSWEBOKCh3)

- abstraction - procedural, data
  "the process of forgetting information so that things that are different can be treated as if they were the same"
- decomposition, modularisation
  splitting up, "usually with the goal of placing different functionalities or responsibilities in different components"
- encapsulation
  "grouping and packaging the elements and internal details of an abstraction and making those details inaccessible"
- separation of interface and implementation *"specifying a public interface, known to the clients, separate from the details of how the component is realized."*
- sufficiency, completeness, primitivity
  "all the important characteristics of an abstraction, and nothing more."

Note crucial role of *interfaces*. This whole family of principles is about fitting very complex software into limited human brains.

Modelling

Let's say: a model is any precise representation of some of the information needed to solve a problem using a computer.

E.g. a model in UML, the Unified Modeling Language. Use case diagrams are part of UML. A UML model

- is represented by a set of diagrams;
- but has a structured representation too (stored as XML);
- must obey the rules of the language;
- has a (fairly) precise meaning;
- can be used informally, e.g. for talking round a whiteboard;
- and, increasingly, for generating, and synchronising with, code, textual documentation etc.

Why design? Why model?

Fundamentally:

Design, so that you'll be able to build a system that has the proprties you want.

Model, so that you can design, and communicate your design.

Both can be done in different styles...

Pros and cons of BDUF

Big Design Up Front

- often unavoidable in practice
- if done right, simplifies development and saves rework;
- but error prone
- and wasteful.

Alternative (often) is simple design plus refactoring.

XP maxims:

You ain't gonna need it

Do the simplest thing that could possibly work

Quote of the day

> *There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.*

C.A.R. Hoare

Reading

Suggested: GSWEBOK2004 Ch3 (see web), for an overview of the field of software design

Suggested: Stevens Ch3, a simple case study; Somerville Ch14 on OOD (and nearby chapters, maybe)