DEPARTMENT OF COMPUTER & INFORMATION SCIENCES

University of Strathclyde Science

Software Engineering

Lecture 5: Design

Billy Wallace
w.wallace@strath.ac.uk

CS993

Lecture Outline

- High-level design (AKA architecture)
  - Frameworks
  - CRC Cards
- Low-level design
  - Design patterns

University of Strathclyde Science

What is design?

- Wikipedia: design is the intentional creation of a plan, process, or specification for the construction of an object or system or for the implementation of an activity or process.
- User interface design is closest to what we usually think of as design, where there is an aesthetic component.
- For me: starting from the requirements, what code will we write?

What else is design?

- UI design is closer to what we traditionally think of as design. We might make drawing of the UI that show what we intend to implement.
- Database design you are covering separately. Once you know the objects you need to store, you can look at how you will store it.
- Don’t rule-out writing code for design. Prototypes are a tried and tested way of designing in other disciplines.

It’s all about complexity

- Software engineering is about building big, complicated systems that need more than one person to build.
- For people to do this, they need to be able to handle complexity.
- The main tool we have to do this is abstraction, and we implement this using encapsulation.

Levels of Abstraction

- Idea
- Requirements
- Architecture (high-level design)
- Design (low-level design)
- Code

Encapsulation

- Functions/procedures - take some code and give it a label. We can now execute the code by calling the label and don’t need to worry about the code.
- Classes - combine data and behaviour and encapsulate these. In this way we also achieve “data hiding”, so we don’t worry about the data (instance variables) either.
- This is why global variables are bad, they have no encapsulation.

Architecture / High-Level Design

- Breaking it down into pieces
- (LLD is putting the pieces together well)
- Assuming we are doing object-oriented design, we are trying to identify objects here.
- In large software projects, you expect senior developers to be responsible for architecture, since it is something you get good at with experience.

Frameworks

- Luckily many complex apps can be built using frameworks that already define the architecture.
- Mobile apps, web apps, Unity for games, etc.
- We used such a framework “Spring Boot” in our first practical.

Common Architectures

- Similar to the design patterns we'll see later, there are common architectural patterns we can use.
- Client-server
- MVC for GUIs
- Multi-tier: presentation, business logic, model.
- The whole idea of a pattern language came from an architect called Christopher Alexander.

CRC Cards

- Originally a teaching tool, but some people like to use it for real.

- Class
- Responsibilities
- Collaborators

CRC Cards

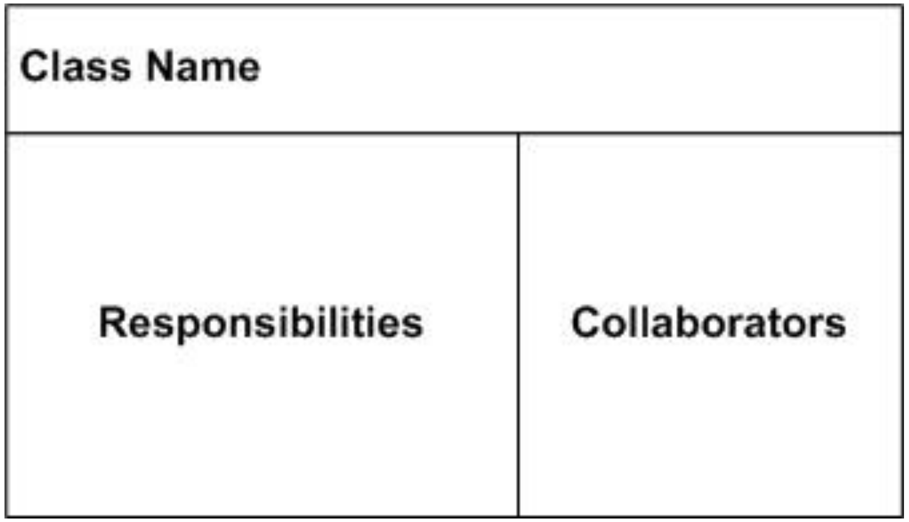

Image from agilemodeling.com

CRC Cards

- We’ll use these for our lab on Thursday, so have a look for decent descriptions online and YouTube videos before then.
- Google is your friend here, but I didn’t see too many good videos when I looked.

Desirable properties of classes

- **Cohesion** refers to how closely all the routines in a class or all the code in a routine support a central purpose (Steve McConnell)
- **Weak Coupling**: If two modules communicate, they should exchange as little information as possible (Bertrand Meyer)
- **Small size** - big things are complicated (Billy)

Low-Level Design

- How do we put the pieces together well?
- Again, we get better at this with experience, but can we can leverage the experience of others by looking at Design Patterns.

Singleton

- How do we ensure that there’s only ever one of something?
- This pattern is a singleton. Search for “Java Singleton” to see the best way to code this.
- This is useful for concurrent systems.

Builder

- How do we separate the (complex) construction of an object from its other functionality?
- Again, this is useful for concurrent systems, especially when used to create immutable objects.

Diagramming

- The most well-known diagrams used in software are UML diagrams. Tools such as IDEs can draw these directly from the code.
- Diagrams are useful, but in my opinion these are not understandable to non-techies and so are less useful than they could be.
- These are useful for helping techies get an overview and understanding code.
- I prefer tools like Evolus Pencil (free) or Microsoft Visio (expensive). Pencil allows you to draw diagrams like flowcharts, but also to quickly draw user interfaces.

Summary

- Design is what we do to get from requirements to something we can build.
- We split this into high-level and low-level design.
- At the high-level, we break things up into pieces and should still be able to communicate this to non-techies.
- At the low-level we should be ready to write code and can’t really convey this to non-techies.

Moving on from here …

- In the practical this week, we'll look at CRC cards.
- For your group project, have a think about "patterns" or architectures that might help.
- Get yourself to the point where you can start to implement code.

University of
Strathclyde
Glasgow