Lecture Notes, CS345, Winter 2018

Introduction to Design

Design

Here's a simple design problem: How do you build a program to play a game?

What would you do first?

> Answer: Ask questions. What game? What kind of user interface? ...
>
> This is called "Requirements."

What would you do next?

> Answer: Do you know how to do what needs to be done? For example: Play chess
>
> - How do you do that? Do you know how to write a program that can choose a reasonable move? What does "reasonable" mean?
> - Do you know how to do a graphical user interface?

Finally:

1. What packages/functions/types etc. will you create? These are programming language and program deployment artifacts. In the future we will use the term components as a generic term for these kinds of things.
2. How are the programming language artifacts organized into a finished program?

Design in the Abstract

Mimicking the previous item: In order to design software you need to

1. know what it is the program is supposed to do.
2. know how to do you do what's needed. Know whatever algorithms and other knowledge you need.
3. decide what programming language artifacts and/or software components you will create.
4. decide how you will organize those artifacts. How will the components fit together?

A big example

You are going to design/build a Web Browser.

1. Some example questions:
    - What protocols other than HTTP/HTTPS will be offered?
    - What kind/level of security protection will you do?
    - What kind/level of anonimity will you allow/provide?
2. You need to know how to:

a) do HTTP

b) parse HTML

c) convert HTML tags into a displayable web page

d) ...

3. How do you organize your program into components? How do those components interact with each other? This is questions two and three, above, combined. Once you've answered this question you will know what you will have to build and how the components are organized.

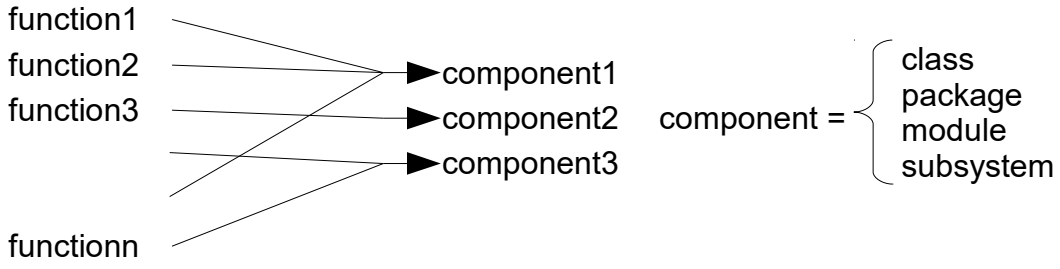
Components Revisited -- Responsibilities

What components do you need?

There are a lot of bits and pieces of functionality required in typical software. For example, in the web browser:

- handle user input
- communicate with external web sites
- interpret HTML
- displaying the page
- ...

Which of the functions go with which other functions? Here's an over-simplified, high-level picture:



And how do the components communicate? Function call? Something else?

Some terminology:

- *Responsibilities*—the functions, data, operations, services, etc. that a component must provide.
- *Interfaces*—how *clients*—other components—make use of the component, how other components get the given component to fulfill it's responsibilities.
- *Implementation*—how the component responds to clients and fulfills its responsibilities.

What is Good Design?

Good design is whatever gives us good software.

What is good software? Here's a laundry list of (just a few) potentially important things:

- Understandability—How easy is it to understand what's going on and how it works?

- Modifiability, Extensibility—How easy is it to fix bugs, add features?
- Testability—How easy is it to test?
- Portability—How easy is it to move between machines? operating systems?
- Performance—How fast does it run?
- Reliability and Availability—Does it work correctly? How long can it run without failing?
- Reusability—Can the software be used in other programs?

Note: Functionality—does it do what it is supposed to—is not included. That is the minimum expectation for any software.

How do we do Design?

Assume you know all the pieces that are needed, how do you decide which pieces of the program are part of which components?

An unworkable answer: Try all the combinations and see which one works best.

- Why is that unworkable? Because the number of combinations is so large that exploring any more than a miniscule portion of the options is not feasible.

A possibly workable answer: Try some combination. If it's good enough, use it. Otherwise, adjust something (what didn't work?) and repeat until the solution is good enough.

Design: Heuristics and Patterns

Real answer: A combination of the above (try alternatives until it's good enough), and

- *Heuristics* (rules of thumb)
- *Design patterns* (previous good solutions to similar problems)

This class will focus on heuristics and patterns.

Where do heuristics and patterns come from? Answer: Experience!

Heuristics and patterns are the result of programmers, you and others, having tried to solve similar problems in the past that either worked (that's an idea to use again) or didn't work (don't do that again).

The things that worked are abstracted (generalized) into rules of thumb and patterns that are applicable to multiple problems.

Final Notes

Experience plays a big role in design. You can't design a complicated piece of software without all of

1. A good understanding of the problem (the requirements.)
2. A good understanding of the technologies underlying the problem. (Browser example: HTTP, HTML, ...)
3. Knowledge of heuristics and patterns that are applicable to the problem and to the technologies (programming language, ...) you are using to solve the problem.

So, in this class you will be asked to learn about heuristics and patterns.

In fact, this class is asking you to do something hard: Learn from someone else's experience.