CMPSCI 187 Fall 2012: Lecture #2

Software Engineering Overview
David Mix Barrington
7 September 2012

Software Engineering Overview

- Software Versus Programs
- Software Engineering Versus Engineering
- Goals for Software Quality
- The Software Life Cycle
- The Waterfall Methodology
- Agile Methods of Software Engineering

Software Versus Programs

- What is **software**? How is it different from CMPSCI 121 or 187 programs?
- Wikipedia: **“Software is a set of programs, procedures, algorithms and its documentation concerned with the operation of a data processing system.”**
- A single program solves a single data processing problem, with specified input and output. The programs in a piece of software may have different users, different desired behaviors, different versions, etc.
- Note that “software” includes “procedures” and “associated documentation” -- everything in the system but the hardware.
- Examples: SPIRE, Mac OS X, *Civilization*, Facebook, the internet protocol...

Software Engineering Versus Real Engineering

- Engineering is the application of science and mathematics to affect reality, particularly by constructing artifacts.
- If you wanted to build a bridge across the Connecticut River from Hatfield to Hadley, a civil engineer could give you a variety of designs and cost and building time estimates for each.
- “Software engineers” apply science and engineering to build artifacts, but they are very bad at estimating cost or completion time.
- Real engineers usually apply known techniques and materials with known characteristics -- software engineers not so much.
- Breakthrough software, like Facebook, changes the environment it exists in.

Goals for Software Quality

- DJW in section 1.1 say that software should (1) work, (2) be modifiable without extensive time and effort, (3) be reusable, (4) be completed on time and within budget.
- Included in “it works” are **correctness**, **reliability**, and **efficiency**. These three aspects will be our main concern for the programs we write.
- Quality is in the eye of the **stakeholder**: maybe a paying customer, maybe not. Developers and maintainers care about the internals of a piece of software, while users care only about observable behavior.
- In this course we are concerned with particular **tools** for building software, the reusable systems for storing information called **data structures**.

More on Software Quality Goals

- A program is **correct** if it meets its **specification**. (In CMPSCI 320 you will find that getting to a useful specification is at least as hard as creating software that meets the specification.) We use a combination of **testing** and **analysis** to determine whether a program is correct.

- A program is **reliable** if it avoids **software failures** -- “unacceptable behaviors that occur within permissible operating conditions”. Absolute reliability is usually impossible, particularly when the environment includes varying conditions of use and even malicious attempts to deny service or break data security.

- A program is **efficient** if the quantity of **resources** it uses, most importantly time and memory, is as small as possible. One of the goals of this course is to begin the study of **computational complexity** -- the science of how resource use, particularly running time, increases with input size.

The Software Life Cycle

- A piece of software, at least metaphorically, is born, lives, and dies. We can identify a number of stages through which most pieces of software go:

1. Problem analysis
2. Requirements elicitation
3. Requirements specification
4. Architectural and detailed design
5. Implementation of the design
6. Testing and verification
7. Delivery
8. Operation
9. Maintenance
10. Obsolescence

- As I mentioned, most of CMPSCI 320 is spent on steps 1-4.

The Waterfall Methodology

- The classic methodology for software design is called the **waterfall approach** because these steps are carried out in order, with the completion of each step producing a carefully documented set of instructions for the next step.

- The first step gives a complete statement of the problem, the next gives a list of the requirements, the third gives a formal specification of the requirements, the fourth gives a description of the parts of the system and how they will interact, and so forth.

- The maintenance process during operation may determine that changes are needed. Where those changes must start will vary -- perhaps the problem can be solved by reimplementing something in the original design, but you might have to change the architecture or even the requirements or the problem statement itself.

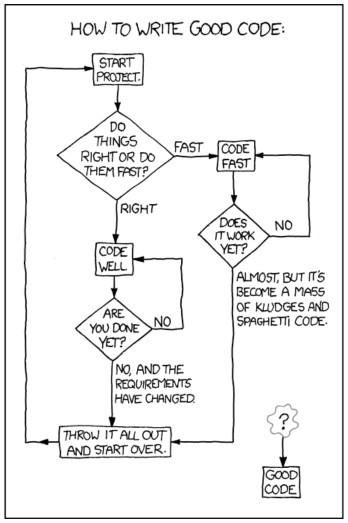
- Those changes “cascade downward” to the later steps of the process.

Agile Methods for Software Engineering

- The larger a piece of software, the less well the waterfall approach works.
- One alternative is the “spiral model” (DFW page 6) where the activity is divided into “setting objectives”, “risk assessment”, “development”, and “validation”, and the design process cycles through each of these four stages in turn as the software gets larger and more useful.
- If the software is going to do something completely new, it may be impossible to specify what it *should* do until some version of it already exists -- a **rapid prototype** that accomplishes some subset of what the final version’s goals.
- **Agile methods** of design **involve the customer** at all stages, deliver the product **incrementally**, are **open to change** in specifications, and use **pairs** of collaborating programmers.

Obligatory xkcd Commentary

- If you don't regularly read **xkcd**, a web comic written by Randall Munroe, you should.
- He is interested in many things other than computer science, but he is very good at getting to the essence of a point in a funny way.
- There are now 1103 of them (three per week for several years) and the backlist provides a wonderful opportunity to be distracted from work.


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE