CSC323 2010S *Software Design*

Class 19: Architecture

Held: Tuesday, April 13, 2010

Summary: We consider techniques for architecting a larger system.

Related Pages:

- EBoard.

Notes:

- Don't forget that you should have completed at least six hours of documented work this week before Thursday's class.
- Reading for Thursday: HFOOAD, Chapter 8.

Overview:

- About the Class.
- Architecture, Summarized.
- Python Style and Beyond.
- Representing Worlds.
- Architecting Our Game of Life.

About the Class

I got a few notes expressing severe frustration with the class (or with certain aspects of the class), so I thought it would be worth spending a bit of class time exploring these issues.

- I will give a commentary on why I think Software Design seems frustrating for the first fifteen minutes of class.
- You will have fifteen minutes to discuss, ask questions, debate, whatever.

So, let's get started.

- Software design is different than most CS classes because the primary outcomes are fuzzier.
  - We are often considering *subjective* issues, such as "is this an appropriate design" rather than *objective* issues, such as "can we prove this algorithm correct" or "is the asymptotic running time of this algorithm acceptable"?
  - We are focusing more on *processes* than *algorithms* (or even code).
- In the end, my goals for the class are therefore that you have practice with these subjective issues and processes.
  - You need to be able to look at code and designs and consider what is good about a piece, what is bad about a piece, and what alternatives there might be.

    - You also need to be able to find the core “ideas” of any piece.
    - It doesn’t matter whether the piece is beautiful or ugly or whatever. You need to be able to do it.
- It is also important how you present your own code and designs to others.
    - Again, you can learn from good presentations and bad presentations.
- One of the big strengths of the Head First book is that it reminds you that central to the process are a number of dialogs
    - And it’s clear that you folks need practice talking about code
- One complaint that I got was of the form “Normally, we learn something as follows: We read about it. Then we hear a lecture about it. Then we do a lab about it. Then we do a homework assignment on it.”
    - This is a 300-level class. I expect that you don’t need that hand-holding by this time.
    - When you don’t understand something, you should have it as a habit to try to code something up to understand it.
    - And I did tell you that the first week of class.

A few of today’s questions really got to the heart of the kinds of things I want you to “learn” from this class.

- “The book suggests breaking large problems down into smaller problems, which certainly makes sense. However, I found that occasionally when I tried to conceptualize flexible designs (creatures in our project, tiles in the book’s project), occasionally I’d be so focused on making the design flexible that it was difficult to actually write a design - i.e. "let’s think of all the possible ways someone could change this before writing down any version of it". In general, can we write a basic version of something first and then expand it to be flexible, or does that lead to moments when you may not be achieving goals the best way because you’ve already locked yourself into an implementation?”
- “Now that I’ve had a chance to work with UML, I’ve noticed that it does not present a complete picture of how a program should work. It lays out the class structure, but does not include much information about how stuff actually "works". How core algorithms should be implemented appears to be an important component of a design. The core component of our design project is the algorithm for actually "cycling" the board. This is a complicated algorithm that involves a large amount of information flow through the program. Shouldn’t the Head First book be focusing on this second aspect of design? Am I just interpreting what "design" means incorrectly?”
- “One thing the book didn’t cover was our group situation, in which you have multiple groups working on the same task and occasionally sharing information. In a real-world setting, if this were the case, would strict adherence to the UML diagram guarantee that our code worked together?”
- “On page 364 the authors make the claim that "Sometimes the best way to write great code is to hold off on writing code as long as you can", this is at odds with the Extreme Programming philosophy of "program a lot to see what works best". Are these two approaches reconcilable?”

There was even a note on a design question that we will explore a bit later:

- “Real Python programmers don’t use getters and setters. Why are we?”

Of course, we want to learn some “concrete” things related to these processes and perspectives.

- UML is one such tool.
- Python is a tool in the sense that we need a common language in which to program.

This isn't to say that I haven't screwed up in some ways.

- I agree with you that the Head First design book is not very good. As I've told you before, in the last offering of CSC 323, students really liked it.
- I have not given you enough feedback
- I have been vague about grading practices
- I deviated too much from things I know and like in this class (particularly focusing on design patterns)

Some clarifications:

- How will we be assessed on the project?
  - If you do the requisite time, you will receive at least a B.
  - If you do not do the requisite time, you will receive a lower grade.
- How will we earn a grade higher than a B on the project?
  - At each stage, I will provide a preliminary grade on each group's code.
  - Because of the power problems last week, I will not grade last week's code.
  - I will also ask your groups to divide up some "extra credit" points.
- What will the final look like? Here are my assumptions:
  - One part of the final will ask you to read and criticize some code
  - One part of the final will ask you about the design of the final version of the project
  - One part of the final will ask you to design a simple system
  - One part of the final will ask you to draw UML for a bunch of code
- Why aren't we doing X part of the project?
  - Because experience shows that you're better off being ready to discard parts of the project.

Now it's your turn. What do you want to discuss about the class.

Architecture, Summarized

What were the primary lessons of these chapters? [Goal: 20 minutes]

- I don't care what you liked or didn't like. I want to know what you think the authors thought the primary points were.

We should also visit the question of high-risk first vs. small first [Goal: 10 minutes]

Python Style and Beyond

There was a request to discuss good and bad coding habits in Python. We'll spend fifteen minutes doing so.

- “Real Python programmers don’t use setters and getters. Why are we?”
  - Because we’re not yet real Python programmers?

Architecting Our Game of Life

Goal: Remaining Time.

- Let’s try architecting our game of life project, using some of the ideas from this chapter.
