Software Construction

Introduction to UBC CPSC 210: Software Construction

Software Systems

Every minute of every day, your life is being affected by one or more software systems. Perhaps you have a digital watch on your wrist whose time display is driven by software. Perhaps you have been on a bus, in a car, or on a plane today, each of which includes many software systems used to help control the vehicle. Perhaps you have turned on the lights in your room, which requires electricity regulated by software systems. Perhaps you have played an electronic game, used a computer, watched television, used a cell phone, texted a friend, written a blog entry, tweeted an event or performed any number of activities that interacted with software systems.

All of the examples mentioned above involve software systems comprising thousands, and often millions, of lines of source code. The act of creating and sending a text message requires software to recognize what you type, convert it into a suitable data format, contact the nearest cell tower, transmit the message, find the intended recipient, who may be on another network, potentially store the message if it cannot be delivered immediately, and finally deliver the message. Some of the software involved in this process is on your phone, some is on the cell towers, some runs at the network provider and some is on the recipient's phone. When you consider that you can text someone and have the message delivered around the world, it is truly phenomenal that such a complex set of interacting systems has been built and works reliably.

Concepts

In your introductory computer science course, you learned fundamental concepts about how you can express computation using a programming language. In this course, we will build on those fundamental concepts to examine how more complex software systems are built. We will look at how various forms of *abstraction* enable software developers to manage the complexity of constructing a software system. We will discuss how *modularity* makes it possible to *compose* systems out of building blocks. We will look at techniques for ensuring the software components that are built are *robust* so that the constructed system has desirable properties such as stably running for long periods of time.

Models

We will use various kinds of models—control-flow models, data-flow models, class diagrams and more—to help design and understand software systems. Each model abstracts one or more properties to help us, as software developers, focus our attention and reason about a part of the computation that forms the system. In this course, we will often explicitly represent the model, often by drawing a picture or writing text. By the end of this course, you will be able to form the models and reason about them in your head.

A software system is itself often a model of something in the real world. Part of the source code comprising the system may model data that must be manipulated, such as a person's health records. Part of the source code may model a computation that must be performed, such as determining the person's eligibility for new glasses under a vision plan.

In most cases, you should be able to distinguish between a *model of the software* and the *software as a model* because we will use particular modifiers, such as control-flow model, to mean the former. When the meaning is unclear, please ask!

Skills

Constructing useful, robust software takes both an understanding of the concepts we will cover and skill to successfully express desired computations in a programming language. As systems become more complex, different parts of the system are expressed in different programming languages and the role of tools to help in the expression and correction of the software grows in importance. In this course, we will primarily use the Java programming language, which may be different than the language you used in your introductory course. We will introduce skills to help you learn how to learn Java (and other languages) and which will help prepare you for your future interaction with other languages. We will also introduce and practice skills for debugging, testing, refactoring and others within a state-of-the-practice development environment (Eclipse) to help prepare you for constructing software systems of moderate size.

Software systems are typically built by groups of people working together. A software development team can work effectively and efficiently only if they are able to communicate with other team members. To help you gain these skills, you will be asked to describe the software you are building both verbally and in written English.

Course Format

The first part of this course will focus on demonstrating fundamental concepts by reading and working with existing small software systems. The initial systems you work with in class and in the labs will be relatively small, but we will ramp these systems in size so that by the end of the term you will be working with systems comprising tens or more of interacting modules. Our focus in the initial part of the course will be *analysis*—learning how to read existing code, extract models and reason about an existing system. By focusing on analysis first, you won't need to jump in and write lots of Java code to start with, but rather you can become familiar with how code is expressed in Java by reading

how others have expressed source code to solve a particular problem. You will be writing some small Java programs as part of weekly programming exercise during this time to give you some practice, but the main focus will be on reading and understanding existing code. In the second part of the course, our focus will shift to *construction* where you will begin to write code that fits into an existing system. The final part of the course will focus on a synthesis of analysis and construction where you will be designing code to solve a particular problem in the context of using existing code libraries. The last part of the laboratory section of the course will stress this synthesis as you will work to extend a non-trivial, and we hope interesting, system.

For more details about the course workload and schedule, please refer to the course web page.