Object-Oriented Design I

Design is a critical step of constructing a software system. Good designs solve the problem at hand and exhibit desirable characteristics, such as being evolvable. Learn how to:

- ✓ **Extract the design of an existing system and record it using a UML class diagram**
- ✓ **Read designs that are (partially) expressed using a UML class diagram.**

What is Software Design?

Given a description of what a software system is intended to do, the activity of software design involves selecting and organizing data abstractions and algorithms to provide a system that does what is intended. We refer to activities related to determining and describing *what* the system is to do as *software requirements* activities. We refer to activities related to determining *how* the system can achieve what the requirements describe as *software design* activities.

Software design is not an exact science. There exist different *guidelines and methods* to help a software developer design software given requirements for the system. There exist different *design principles* that a software developer aims to achieve in a design. In the coming lectures, we will look at an approach to help guide object-oriented design, called the responsibility-driven design approach. We will look at ways to help write down a design, including UML class diagrams. We will look at principles we want to achieve in designs, including low coupling and high cohesion. We will consider good design patterns that can be used to help achieve certain qualities in the designs you create.

Along the way, we will look at the implications of design on implementing a system (i.e., actually writing the code and making the desired system function) as the first criteria a software design must meet is that it can be implemented. Some designs may exhibit fantastic flexibility, allowing new features to be added easily. However, implementations of the design may function slowly. A good software designer must be able to make such trade-offs, always considering whether or not a design can be implemented with desired characteristics when the system executes.

One thing to keep in mind as we look at various aspects of software design is that design is an iterative process. For simple problems, great experienced designers may be able to design a system well in one try. In general, even experienced designers will write down initial ideas about a design, refine those ideas, change them, start over, design some more, implement part of the system, return to design and so on. A hugely important part of design is the willingness to try different ideas, evaluate them and try again.

Capturing a Design: UML Class Diagrams

One approach to initially capture an object-oriented design is to record it on blank sheets of paper or index cards. This approach can work great for the initial stages of software design and initial stages of refining that design. When you want to describe a design in more detail to share with members of a team or more precisely analyze characteristics of a system or even generate code directly from a design, a more complete and formal representation is needed.

The most popular notation used to capture designs is UML (the Unified Modeling Language). UML includes many different notations for capturing different aspects of a system. We looked at one part of UML in the Control and Data Models reading, UML sequence diagrams. In this reading, we consider UML class diagrams, which help capture the structure of an object-oriented design by describing the classes involved in a system, how the classes relate and how objects of the classes relate. A UML class diagram does not directly depict how the system works, but the structure of the classes can be used to help developers talk through how the system will function. If a developer wishes to specify how objects of the classes will interact when the system functions, UML sequence diagrams are used. It is not uncommon to see software developers sketch designs using various short-hand forms of UML class diagrams.

References and Further Reading

As there are a number of good descriptions of UML class diagrams on the web, I have not described the details in this reading. Please read the article linked below. You can skip the sections on Association Class and UML 2 Additions:

UML basics: The class diagram: An introduction to structure diagrams in UML 2 by Donald Bell, IT Architect, IBM.
(http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/)