CS122 Lecture: Reuse, Components, Frameworks and APIs

Last revised April 7, 2015

Objectives:

1. To introduce the basic concept of re-use
2. To introduce the notion of component-based software engineering
3. To introduce the notion of a framework

Materials:

1. Schach §8.3.1 and §8.3.2 to read
2. Example of a component-based system from Wikipedia
3. Javadoc for ocsf package
4. Demo of framework based client-server chat system to run - including executable of client on a USB stick
5. Projectables of code for server and client for chat system
6. Demo of framework based client-server address book system to run - including executable of client on a USB stick
7. Projectables of code for server and client for address book system
8. Ability to demo man command
9. Ability to project Java SE API and java.sun.com list of all API's

I. Re-use

A. One idea that has been prominent in software engineering is the idea of re-use. One textbook defines this idea as follows: "Reuse refers to using components of one product to facilitate the development of a different product with different functionality" (Schach 2008 p. 216).

1. The idea is actually quite old - some of the initial work goes back 40 years.

2. But reuse has been of particular interest in connection with object-orientation.

B. Re-use is desirable for several reasons

1. By re-using previously developed components, rather than developing new components from scratch, it becomes possible to develop a solution to a new problem more quickly and inexpensively.

2. Further, a reused component is likely to be more reliable than one developed from scratch, since it has already been tested in one or more previous uses.

3. Developing components with re-use in mind typically results in components that are better designed, documented, and tested.

C. Re-use can occur at many levels

1. Portions of code may be reusable. In particular, in OO it is possible to develop reusable classes

2. But so are things like portions of a design or documentation, etc.

3. As one important example: design patterns are reusable. Indeed, the subtitle of the "Gang of Four" book is "Elements of Reusable Object-Oriented software".

D. It has been estimated that only about 15% of a typical software product serves a truly original purpose. Thus, up to 85% of a piece of software may be constructed from re-used components - though in practice 40% seems to be an achievable upper bound. (Schach 217)

E. If reuse is a good idea, why isn't it practiced more often?

1. It is more expensive to develop a reusable component than it is to develop something that is intended to be used in just one place - in terms of careful design, documentation, and testing.

2. The “Not Invented Here” syndrome. (If someone else developed it, it can’t possibly be as good as something I could produce)

3. Related to this is the reluctance of developers to trust work done by others - a fear that something that is reused might introduce a fault into the product.

4. There is also the problem of maintaining libraries of reusable components in such a way as to make it easy for someone to find what they’re looking for.

F. Two case studies - one illustrating benefits of reuse; the other dangers.

1. Read Schach §8.3.1

2. Read Schach §8.3.2

II. **Components**

A. An idea closely connect to reuse is the notion of a reusable component.

Perhaps the idea can be illustrated by looking at some examples of the notion of a component from outside the realm of software engineering.

1. Standard hardware fasteners

a) If you go to any hardware store in the country, you’ll find a variety of screws and other fasteners.

b) Though there is great variety, there is also a definite structure. For example, machine screws

(1) come in many lengths

(2) can be made of a variety of different materials (steel, brass, nylon)

(3) Can have several different head styles (flathead, roundhead, oval heat) and types of slot (straight, Phillips, Allen etc.)

(4) But they all have one of several standard thread types - e.g. 6-32. Moreover, a 6-32 nut will fit any 6-32 screw regardless of length, material, or head style.

c) When a manufacturer designs a product, it normally uses one of the standard designs instead of creating a custom design just for that product.

2. A home stereo system

a) My stereo system at home consists of a number of component parts, purchased over a period of several years.

(1) A Sony CD changer

(2) A Sony receiver/amplifier purchased several years later

(3) A Panasonic DVD player

(4) A pair of Bose loudspeakers

(5) Several Jensen loudspeakers

b) Though these components were purchased over a period of many years and come from four different manufacturers, they all work together perfectly. In fact, even the cables that connect them are all (except for the speakers) the same type of cable - standard HiFi cables with RCA plugs on both ends.

Why is this possible?

ASK

This is possible because of well-established standards in the industry concerning connectors, signal levels, impedances etc.

B. There is an approach to software engineering - called component based software engineering - that focusses on developing systems by combining components. The complete system must then include some "glue" code that holds the components together (analogous to the cables used in a stereo system) - but the bulk of the functionality is provided by the components.

   Example: PROJECT Example of a Component-based System from Wikipedia

C. While the full vision of software developed by interconnecting "standard" components is far from reality, there are a number of places where this sort of approach is used in practice.

   1. Standard libraries (the Java Class library , Microsoft .NET, libraries of scientific computing functions, etc.)

   2. Libraries supporting specific applications (Java EE, etc.)

   3. Plugins for web-browsers and various software packages

III. **Frameworks**

A. One interesting approach to re-use centers on the notion of a framework. A framework is a skeleton of a particular type of application that can be converted into a complete application by adding a few key elements of code.

B. Perhaps the best way to understand the concept is to look at an example. A book that was once used for a previous version of this course included an example framework for creating a simple client-server application.

1. The framework consists of two packages - `client` and `server` - used to develop the programs running on the client and server system, respectively.

   a) The main class in the client package is an abstract class known as `AbstractClient`. As a minimum, to create a client application, one creates a concrete subclass of this class which implements just one method: `handleMessageFromServer()`.

   b) The main class in the server package is an abstract class known as `AbstractServer`. As a minimum, to create a server application, one creates a concrete subclass of this class which implements just one method: `handleMessageFromClient()`.

2. Using this framework, it is possible to create a simple client-server application by writing a minimum amount of code.

   a) DEMO:

      Run both Chat server and client on my system and demo. Then ask one or more other students to run client (from USB stick) on their systems and demo

   b) How much code do you think is needed to accomplish this?

      ASK

      PROJECT Code for Server and Client - Focus on code found in `handleMessageFromServer()`, `handleMessageFromClient()`.

      Note that the complex networking code is all found in the abstract base classes re-used from the framework, so these classes only need to deal with the application-specific matters

3. This framework can be used to create many other kinds of simple client-server application.

   a) DEMO: Run both AddressBook server and client on my system and demo.

   b) PROJECT: Code for AddressBook server and client

   c) (Of course, a database might be used for storing the address book and the response to an inquiry would be more meaningful - this example is just to illustrate the power of the framework)

C. Framework code includes three kinds of methods that are relevant to a person using a framework. We will illustrate each using the server class of the chat system.

   PROJECT: Code for class `SimpleChatServer`

   1. Slot methods

      A slot method is one that is typically left as abstract in the framework (and thus is a method of an abstract class). To use the framework, one must supply a concrete implementation of the method, which does whatever the specific application requires.

      Example: the method `handleMessageFromClient()` in class `SimpleChatServer`.

      (Show how abstract in documentation for ocsf plus code in `SimpleChatServer`.)

   2. Service methods - the methods provided by the framework that the user can use in coding the slot methods.

      Example: `sendToAllClients()` in `handleMessageFromClient()`. Compare this to the address book example, which uses `sendToClient()`, which to send its result only to the client that requested it, as called for by this application.

3. Hook methods - methods that allow the user to add additional functionality if appropriate for a specific context. These are methods that are typically implemented as null methods (i.e. a body consisting only of {} or returning some default value in the framework. But the user can override such a method to add application-specific functionality where needed.

   Example: the ocsf server framework class `AbstractServer` provides a hook method called `clientConnected()`, which is called whenever a new client connection is made. The default implementation is empty, but a concrete implementation can override this if it wants to do something special when a client connection is made.

   DEMO: Open server project in NetBeans.
   Add the following code to `SimpleChatServer`:

```
protected void clientConnected(ConnectionToClient
client)
{  System.out.println("Connection established to " +
        client.getInetAddress());
}
```

   Run project with added code, then run client and note how message appears at server when client starts.

D. Basing an application on a suitable framework - rather than developing everything from scratch - can not only save a lot of effort, but also can result in more robust code, for two reasons:

1. The framework developers have (hopefully) produced code that is more carefully designed and thoroughly tested than what the application developer might produce for just a single application.

2. By application developer can concentrate on the core functionality of the application, without having to become enmeshed in the details of functionality (like - in the case of our example - network communication) which is not central to the application.

E. However, developing a quality reusable framework can be much more expensive than developing single-application code, because of the need for generality.

1. Example: show hook methods of `AbstractServer` in the javadoc. The designer of a framework needs to anticipate the things an application might want to do and provide hooks for them, as opposed to simply thinking about the needs of a single application.

2. However, purchasing a suitable general framework from a vendor is often much cheaper than developing one's own application-specific code.

IV. **API's**

A. The capabilities of a re-usable software component or set of components are typically specified through what is known as an API - Application Programming Interface.

B. The idea of an API originated with operating systems. The capabilities furnished by an OS to applications running on it would be specified by its API.

1. The standard API for Unix-like systems is specified by an IEEE specification known as POSIX (Portable Operating System Interface). One important part of POSIX is a set of specifications for system routines that can be used by application programs.

   a) Example: man getuid

   b) Many Unix-like systems support other API's as well

2. Likewise, there is a Microsoft Windows API specified by Microsoft.

3. There is distinct no Mac OSX API per se, because OS X implements both the POSIX API and Berkeley Unix APIs; however, there is an API known as Cocoa that supports building GUI applications on Macintoshes.

C. In addition, it is common to specify the set of services provided by any reusable component by an API.

1. In the case of Java, an API is specified using Javadoc.

   a) You are familiar with the API for the standard java library.

      PROJECT; walk through

2. The Java platform includes quite a number of other API's
   DEMO: go to java.sun.com; follow link for Java API's (under Essential Links on left side)

3. The Microsoft .NET framework is supported by Microsoft on Windows platforms, but there is also an open-source implementation known as Mono which is intended to provide equivalent functionality on other platforms.

4. Commercial libraries or frameworks will typically be specified by an API as well.

D. When specifying an API for a language-specific component (e.g. a library or framework), a suitable language-specific system may be used (e.g. Javadoc). But API's can support multiple languages, in which case the API is specified in a language-neutral way (e.g. by using a language-neutral interface definition language (IDL))