CST0006 – Computer Programming Foundations

INTRODUCTION to WEEK11

In this lecture and lab section we are going to finish up with some extras tools and concepts that allow you to understand and develop better software. There is a seemingly endless tools and rules to learning how to build better software. The more you learn, the better equipped you will be.

After this lecture and lab section, you will have a general understanding what tools and libraries that are available to you as a programmer.

Software Libraries

A collection of pre-written software available to you to add functionality to your program. Software libraries are supported and included by the programming language that you are using, and have been test-driven and hardened against software bugs. You can choose to include these libraries into your code instead of recreating the wheel every time you write a new program. The better you know what is available to you, the more functional you become as a programmer.

Frameworks

Very much like a library, but acts like a extra functionality that you conform to. Frameworks make the rules and hold up the hoops that you need to jump through. Most design decisions have been made for you. GUI software tools are considered frameworks. You can include them into your code, but there are strict ways that you need to implement them into your program. The framework makes up most of the design architecture and you fill in the blank parts.

Both software libraries and frameworks are considered modules, as they are separately written modules. When modules are grouped together under one directory, they are considered a package.

Application Programming Interface (API)

An interface is described as the point where two system meet and interact with each other. When you're driving a car, you are given a steering wheel, lights, a gas pedal, a break pedal… these are the tools that you use to interface with your car. For your radio, you get an on/off button, a tuner dial, a volume dial and some buttons to save radio stations. An application programming interface is the methods you can call, or your program can call to interact with the application you are interfacing with. Most times you don't get to see the underlying code that makes the software run, you only get the available methods to make the code do its job.

Architectural Patterns

Architectural Patterns are the high-level design of a program. Architectural patterns and design patterns are often described as the same thing, (depending who you ask). Architectural patterns are more concerned with how a large program are broken into the structural pieces and how they relate to each other. Where as a design pattern is concerned with how the classes are broken up and interact with each other.

Operating Systems

Operating systems is a perfect example of how a large project can be broken into different architectures. The Monolithic vs Microkernel "which is better" debate has spanned decades and will probably span even more decades. The monolithic approach is that the kernel is a single large process that runs in a single address space. An example of a monolithic project is Linux. The microkernel approach is when the kernel is broken into smaller services (servers) and some run in the user space. Examples of microkernel systems are GNU Hurd and QNX. An example of a hybrid system would be OSX and Windows.

Model-View-Controller Pattern (MVC)

The MVC architectural pattern is also considered a design pattern. Of all the different patterns, MVC is one of the most popular design patterns. In fact, there are some programming languages that demand you design your programs using this model. The MVC architectural/design pattern allows you to break apart the presentation layer from the data layer so that you can design each part of the program separately. It allows you to separate the data layer from the presentation layer so that you can change each part without affecting the other part. There are an insane amount of ways to interpret this design pattern, but if you can successfully separate each of these three components, you've succeeded in creating the MVC design pattern. Regardless of object oriented programming language used, you can (and should) implement this design pattern in some form or other.

Model	The data. This is the programming that deals with the data. Whether it be a database, a text file, a list of names, a bunch of points on a map. The retrieval and organization of the data falls under this category.
View	The presentation. This is the part of the program that deals with how the data is displayed. It can be written for a web page, the command line interface, a GUI framework, or even a different GUI framework. The presentation deals with the code that you need to build your visual interface.
Controller	The glue. This is the part of the program that joins the data with the presentation. The presentation layer shouldn't know anything about the data layer and the data layer shouldn't know anything about how to data is to be viewed. It's the job of the controller to make calls to and receive data from the model and then push them to the presentation layer to display those changes.

Design Patterns

Design Patterns is a term used to describe the “best practices” way of building or designing your software to solve common design problems. There are many different design patterns to choose from, each design pattern is battle tested ways of building better programs that have been developed over decades of trial and error.

Classification of Design Patterns

Design patterns can be organized into three categories:

Creational	How an object can be created
Structural	How objects are connected and related to each other
Behavioural	How objects communicate with each other

The most popular book on the subject is called **Design Patterns - Elements of Reusable Object-Oriented Software** and was written by four authors affectionately known as the Gang of Four (GoF). As with every other topic, this one is a course all on its own, and I just wanted you to be aware of these concepts. We won’t be diving any deeper than this right now, but once you start learning how to code better, you’ll want to check this topic out further.

GUI Programming

Python has multiple GUI frameworks available for you to chose from. Tkinter is the standard GUI toolkit supported directly by Python. However, like any tool the one you know best is the one that you can use the best. If you are a fan of open source/cross platform wxWidgets, then wxPython is an interface that allows you to work with wxWidgets. Likewise PyQT is available for those that are used to building GUIs in Qt and PyGTK for building GUIs in the cross platform/open source GTK.

Tkinter Window Example

```
from tkinter import *

frame = Tk()

frame.mainloop()
```

Tkinter Adding a Title

```
from tkinter import *

frame = Tk()
frame.title("Awesome Title")

frame.mainloop()
```

Game Programming

Game programming has driven the advancement of computer hardware like nothing else. The push for faster, better and stronger has been funded mostly through the gaming industry. Then every time they build it, the gaming industry absorbs it and wants more!

Gaming Engines

A gaming engine is a framework that bundles things like a graphic-engine for rendering 2D and 3D, animation, light/shadows, physics-engine for simulated gravity, wind, explosions, sound-engine to allow for 3D-like sound, and memory management. Some popular gaming engines include Unreal and Unity. Some popular graphics engines include DirectX, OpenGL and the all new open source, cross platform Vulkan.

Pygame

Pygame is a cross-platform set of Python modules crated for writing video games in Python. It has been released as open source under the GNU Lesser General Public License (GLGPL). Pygame has lots of free tutorials on how to use it and build simple and fun games with.