**Case Diagrams**

Too many times, a customer will state what they want in a program, only to have the finished product be something entirely different. I can understand why it would. I have the “fixer” mentality. Tell me your problem, and I’ll get right to work fixing it. A case diagram provides the foundation a developer needs to prevent this from happening. The program should solve a particular problem, but it shouldn’t be unnecessarily complex or redundant.

**State Diagrams**

According to geeksforgeeks.org: “A state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.” I have seen my progression in this class change like a state diagram. Just as code needs to evolve over time to meet changing needs, I have had to change my perceptions on team development in order to learn and implement acceptable practices.

**Case Documents**

As I reflected on my case doc, I noticed the similarities it has with a syllabus. Just as a syllabus specifies my expectations for this, a case document lays out in report style the flow that the code needs to take to complete its process.

**Sequence Diagrams**

As with all cases, you have to drill down even further to where the rubber meets the road. Plans have to be made, changes effectuated, and details finalized in order to get a project rolling. This semester, I started doing a daily goal and task list. It’s one of the last things I look at before bed, and one of the first things I look at when I get up. It is my daily sequence diagram of the things I need to accomplish, and helps my rubber meet the road.

**Model-View Controller Pattern**

This greatly facilitates the different aspects of a program. Not all facets of a program need to do the same thing. Attempting to have it that way will result in unwieldy code that is incredibly difficult to maintain, especially as it grows larger. As with code it is important that such serration is maintain in life as well. Different divisions in our lives should complement one another, and care should be taken that they do not overrun each other.

**Application Controller Pattern**

Some code needs to be obfuscated. APC controls the flow of information within a program so that proprietary information stays that way. It is not supposed to look pretty, but is specifically for handling business logic. Just because we know something should be done, we don’t necessarily know exactly how it is done.

**Java Collections**

As I studied the various collections, it rapidly dawned on me that not all information is of the same value. It also became clear that looking at the same set of information from a different perspective allows different and more refined conclusions to be made. This is especially relevant when viewing said information, as people must interpret the significance of data, and it obviously cannot be absorbed en masse.

**Threads, Executables, and Runnables**

These three amigos are all about efficiency. It is so handy to be able to do more than one thing at once. However, try to do too much, and the opposite happens; nothing gets done. Example, as I sit here completing this assignment, I get a little tired. I know it’s hard to imagine, but sometimes I get sleepy when doing Java. So I pop open a YouTube tab and play a little toccata fugue in D minor by Bach. It’s just the right amount of creative engagement my mind needs to keep going. But, if it’s binging The Arrow, it would take me forever to get this assignment done. Balancing around available resources is a powerful tool to accomplishing more.

**JUnit Testing**

When I took Object Oriented Programing, I did not care for Unit testing at all. I knew my code worked, I and I saw no need for testing my inputs and outputs. It was just extra code that I did not want to enter. However, as I worked in this team, I visualized that anyone that comes behind me to improve or troubleshoot my code could very easily break it. But if proper testing were put into place and constantly updated any mistakes entered in would quickly be recognized and rectified before they were even implemented.

**System Level Test**

This is another rubber meets the road scenario. Does the code actually do what it is supposed to do? This is especially true for the end user, who will most likely not understand why something doesn’t work, and what steps need to be taken to troubleshoot it. Furthermore, it is incredibly simple to do, and such an important step in a feedback loop that it should never be overlooked.

**Servlets, HTTP, Hibernate and JSON**

I have seen it announced on my preloaders, that over 3 billion devices run on Java. It is the most ubiquitous programming languages and would serve any coder well. Yet we know that it needs to communicate with other systems. In today’s environment of cloud computing, any program that cannot communicate across networks will quickly become obsolete. Servlets are invaluable for running back end web requests, HTTP is and will most certainly continue to be the internet’s preferred protocol for web traffic. The databasing of information and the need to access said information is at the very core of Information Technology, thus Hibernate is a vital means in the flow and view of information. JSON is also a common language that almost all languages can speak, and is simple enough in nature that it can be easily implemented. While uniformity has it’s advantages, it is diversity and flexibility that makes a system strong, whether it be a programming language, or an entire country.