**Design Document**

**Synopsis:**

When I started this assignment I first had to learn Cobol and a bit of history behind it. After I started studying it a few days from the slides given in class and the code, I still felt very confused. I barely understood Cobol after a few days, and I thought of myself as someone that learned languages very easily! I had to really work hard on researching lots of little details with Cobol where I could even understand the basics and start working from the ground up. After a few days I felt like I understood Cobol enough where I could start working on the assignment given. Getting to this stage was much longer compared to Ada and Fortran and I still felt very little confidence in starting this assignment.

When I started reading the assignment, I noticed a few legacy features but still could not grasp the entire code fully as well as I should be able to. After a few hours of trying to understand the code, I decided to write a base of code that I understood and reference the legacy code to work with to build towards. I was not able to understand the Cobol legacy code and knew it would take me forever to fully understand it and it made sense to me to start fresh and reference the legacy code.

I decided to start with reading in a file line by line and next to get a string from this file. Once I got a string from this file I referenced the legacy code to understand the theory behind calculating the roman numerals by iterating through the string. I also followed the rules on the assignment guideline to also help me finish the assignment.

Some of the legacy features that were removed were GO TO statements and the way variables were used are different also. A few features that were changed were the If statements and they follow a new format in the newest Cobol settings. The newer if statements use if condition then //do stuff end-if while the older format does not follow the standard.

Since the legacy code was written to take input from the keyboard, the final result that I wrote will look different compared to the legacy code because we were tasked to read in from a file and I am also not the original author of the code. The theory behind calculating the roman numerals is roughly the same, and the final converted code roughly looks the same as it was referenced from the legacy code the entire time.

**Identify Legacy Features:**

• Older Cobol code us GO TO statements that go to different parts of the program, while newer Cobol versions consider this practice a legacy feature and no longer used. This was fixed from general flow of programming, and if/else statements

•Older versions of Cobol use capitals for everything, newer versions should not use capitals but can use them if they desire.

•Older versions could have an if statement like: IF S(I) IS NOT EQUAL TO ‘I’ GO TO B1. While newer Cobol versions would have: if S(I) not = ‘I’ then // do stuff end-if, the format has changed and it follows a new procedure!

•Older versions of Cobol would have: MOVE 10 TO RESULT. While newer versions do not use “.” At the end of everything and could be move 10 to result as the statement

• Variables in older versions of Cobol would be declared like: 77 I PICTURE S99 USAGE IS COMPUTATIONAL, while newer versions would now look like: 77 I pic X(99). PICTURE clause is changed to pic!

•Older versions have a PERFORM statement but the newer versions have a perform statement and it ends the perform with end-perform to know where it ends! This is a big improvement on getting rid of some legacy features as it is more readable

•Older versions need to have certain amount of spaces to make the code work properly while the newer Cobol versions can compile with certain flags to make that not required!

**Q & A:**

• **Would it have been easier to re-write the program from scratch in a language such as C?**

It depends on the background of the programmer, since being a student in the 2000s we start learning with C programming while legacy programmers might have started with Cobol. Since my background is mostly C it would have been easier for me to re-write the program from scratch instead of converting legacy code.

• **What were the greatest problems faced during the re-engineering process?**

The greatest problem faced was that I had to first learn Cobol for me to even start on the re-engineering the assignment given from prof. Next it was very difficult for me to understand Cobol in the first place, it was very hard for me to grasp after several days of studying it and I still feel like I barely grasp Cobol. The biggest problem faced during the re-engineering process was even understanding where to start and how to start converting the professor’s Cobol assignment as I do not understand Cobol very well.

**• Is your program shorter or longer? Why?**

My program is longer than the Cobol legacy code. Since the original code was written by the author, and I can assume they understand Cobol fully and were experienced with the compiler that their program would be written shorter than mine. Since I am not experienced with Cobol and we are unable to contact the author to fully understand their own program that my program will be longer as I am trying to convert it to a more modern Cobol version.

**• Is there a better way of writing the program?**

When the code is converted from older Cobol versions to latest conversions, yes there will always be better ways of writing the program. Since the code was written in an older Cobol version and it has legacy structures such as GO TO, all caps, variables with different standards, if statements with different standards, and some of these features are no longer used, we can assume that there are better ways of writing the program with newer features. I believe the program was very well written and only needed to update a few of its code for it to work properly.

**My Experience with Cobol:**

Cobol was very hard for me, University has only really taught us mostly C and a little bit of Java / Perl . It was very hard to grasp compared to Fortran, Ada, and I struggled for a long time to even figure out what to do. After days of studying the slides/code given by the professor on the course link, I still did not understand it very well.

After a while of studying I decided to start the assignment and see if I could start understanding it as I go and once I started to try to figure out how to convert the professor's code it made me even more confused and I almost felt like giving up on Cobol all together! With all the legacy features like GO TOs it even made it more confusing for me, I had to start writing a bit of my own code to start this assignment to even understand what is going on.

After writing a base for the assignment by referencing the professor's assignment, I decided to remove a few features and change a few features around so I could better understand everything and get closer to even being able to finish the assignment. I tried very hard to keep the theory behind the legacy code and follow that exactly while changing it a bit so it works with latest Cobol and concepts that I understand also.

Most of the concepts in Cobol confused me and felt very “foreign” to me. Even after I completed this assignment fully, I still feel very confused with Cobol. But once I started to get use to Cobol, it is a very interesting and different “perspective” on programming where it makes me rethink about programming in general for Software Engineering as I approach the future.