CIS 3190
Cobol Data Statistics Reflection Report

James Singaram
0758172

March. 22, 2021

After years of using Java and C and months learning Ada and Fortran, my experience working with Cobol is best summarized as “not bad, just different.” The language’s novel formatting and constraints presented numerous challenges that largely stemmed from assumptions and habits gained from previous languages. Cobol’s strengths shone brighter with time and experience.

The first, major hurdle to overcome was in deciphering the original code. As with previous assignments, the first task was to review and test the initial version to translate it into a flowchart. The provided code was densely written and performed multiple disparate tasks within a single block of code. While this vastly reduced the lines of code, it made the program very convoluted and complicated interpretation. Before a flowchart could be assembled, all actions were recorded and then categorized. Each of these groups would later become a paragraph in the final version. Of particular note, writing lines to the output file was scattered throughout the entire program. By separating these it became possible to contain file output to the very end of the updated version’s execution.

With an outline to follow, writing the modernized version was simplified to two tasks. First a rough skeleton of descriptive paragraph names were created. These paragraphs were stubs that contained nothing more than a display command to indicate what was executing. After all formatting issues and bugs were resolved with this basic version, each of the paragraphs was completed in turn. A specific design objective for this approach was to make the order of execution in the ‘procedure division’ so self-evident that someone with no programming experience would be able to understand what was being done. Paragraphs were kept as short as possible, with most being under five lines. Some were as short as two lines. Paragraphs relating to the output file were longer in order to format the output file correctly.

That the program could be made to execute so plainly speaks to the readability of Cobol. While the upper case used in the original code hindered interpretation, the simple function names and syntax more than made up for it. When written fully in lower case it becomes possible to rapidly review large segments of code. In comparison to C and java the function names are clean and rarely need any further description.’ Compute’ will perform an equation. ‘Move’ will place a stated value into a variable. ‘Perform’ x will perform paragraph x. The presence of basic math functions move, compute, add, and subtract were initially grating, however their strength in clarifying math-heavy segments of code become apparent while working on the statistical analyses in the program. Every line of math was helpfully preceded by helpful red letters (using a lifesaving plugin to colour text for .cob files) that stated what was being done on a given line (see Figure 1).

Despite these benefits, I am uncertain whether I would choose to use Cobol in the future. While I recognize that the limitations and difficulties that the language presented to me may merely be the consequence of my inexperience, they devastated any attempt at forming a favourable first impression. The overall structure of the .cob file introduces significant inefficiencies while working. By separating executable code from variable declaration the program ran on global variables. Not only did this add time to scroll up and double check which variable was being used for a given task, it also increases the chance of errors. I had underestimated the benefits of being able to declare variables within functions until Cobol prevented me from doing so. In a similar vein, variable declaration required the exact format of the variable to be declared. There were no short-hand ints, or floats, or chars. Numbers were spelled out to the decimal point, and if greater accuracy was required then the variable needed to be changed. This presented issues early in the program’s creation with variables that did not align perfectly. Lastly, executing loops required that repeating code was stored in another paragraph. While this is fine for comlex tasks, it added complexity when the looping paragraph was only a couple of lines long.

While I am uncertain whether I would use Cobol going forwards, it presented a new way of crafting programs and the experience will doubtlessly prove valuable in future pursuits.

```
*> Calculates the harmonic mean of the numbers
calc-harmonic-mean.
    move 0 to i.
    move 0.0 to harm-sum.
    move num-records to temp-geo-mean.
    perform calc-harm-sum
        until i = num-records.
    compute num-harm-mean = num-records / harm-sum.
```

Figure 1. Helpful descriptors at the start of each line.