|  |
| --- |
| Johns Hopkins |
| Module 4 Assessment |
| Function Points |
|  |
| **Brian Loughran** |
| **6/20/2020** |

|  |
| --- |
|  |

**Problem Statement**

You are assigned to support the size estimation of another wireless communications product.  You have analyzed the requirements and collected the following data:

|  |  |  |
| --- | --- | --- |
| 10 simple outputs | 5 simple interfaces | 4 simple output inquiries |
| 7 average outputs | 7 average interfaces | 11 average output inquiries |
| 12 complex outputs | 8 complex interfaces | 10 complex output inquiries |
| 6 simples files | 14 average inputs | 5 simple input inquiries |
| 10 average files | 6 complex inputs | 8 average input inquiries |
| 4 complex files |  | 12 complex input inquiries |

*Table 1: Number of outputs, inputs, inquiries, files and interfaces*

All environmental factors are average except as follows:  data communications is significant; performance objectives, operational ease, and multiple site use are strong throughput; and the program manager wants 50% of the code designed for reuse for another project expected to start shortly after completion of this project.

1)  Compute the adjusted function point total.

2)  Compute the equivalent lines of code in Graphic Icon languages.

Show your work.  (8 points)

**Assumptions**

The lectures lay out a 6 step process for computing the number of function points for a particular software project. While I have seen other ways of computing function points (more empirical based), we will use the method from lecture, with all the assumptions of the lecture process built in. For example, the lectures assume specific constants for the weight complexity which may not always be accurate. Further, some of the terms like “simple”, “average”, and “complex” can be considered to be subjective in tallying inputs, outputs, inquiries, etc.

Further, we assume that all of the outputs, inputs, inquiries, logical files, and external interface files were properly accounted for. Missing these as part of the project will result in an inaccurate tally of function points.

To compute to lines of code, we use a factor based on the adjusted function points. This computation, of course, is simply an estimate. Following this process will not yield an exact result, and not too much surprise should be yielded if the actual lines of code for the project vary to an extent from the number of lines of code estimated at the beginning of development.

**Computations**

The lecture lays out a 6 step process for computing the number of function points which we will use for this problem. These 6 steps will be followed in this section to get the total number of function points and total number of lines of code based on the inputs from the problem statement.

**Step 1 – Count Function Points:**

The first step in the process has already been completed for us in the problem statement as shown in Table 1. This has tallied all the outputs, inputs, inquiries, files and interfaces we need to proceed to step 2. This information is compiled into a more helpful format for the following step below in Table 2:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Simple | Average | Complex |
| # outputs | 10 | 7 | 12 |
| # inputs | 0 | 14 | 6 |
| # output inquiries | 4 | 11 | 10 |
| # input inquiries | 5 | 8 | 12 |
| # files | 6 | 10 | 4 |
| # interfaces | 5 | 7 | 8 |

*Table 2: Compiled function points*

Note that since simple inputs was not an item in Table 1, we assume there are 0 simple inputs.

**Step 2 – Estimate Weight Complexity:**

The next step in the process laid out in lecture for computing function points is to estimate the weight complexity. This process basically takes the number of each item and multiplies it by an arbitrary weight value (complex tasks have more weight than simple, weight factors for interfaces are heavier than inputs, etc.). By using what we have in Table 2, we can use the given multipliers from lecture (shown in bold) and compute the total number of function points for the given project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Simple | Average | Complex | Function Points |
| # outputs | 10 \* **4** | 7 \* **5** | 12 \* **7** | 159 |
| # inputs | 0 \* **3** | 14 \* **4** | 6 \* **6** | 92 |
| # output inquiries | 4 \* **4** | 11 \* **5** | 10 \* **7** | 141 |
| # input inquiries | 5 \* **3** | 8 \* **4** | 12 \* **6** | 119 |
| # files | 6 \* **7** | 10 \* **10** | 4 \* **15** | 202 |
| # interfaces | 5 \* **5** | 7 \* **7** | 8 \* **10** | 154 |
| Total: | | | | 748 |

*Table 3: Computed Total Function Points*

Of course, the number of function points in the right column is just the sums of the weighted function points in the simple, average and complex columns. And the total function points are just the sums of each of the rows. Note that to compute the total number of function points, we have to select the greater for the input and output inquiries. This is specified in lecture.

**Step 3 – Evaluate Environmental Factors:**

The next step is to evaluate environmental factors. According to the problem statement, all environmental factors are average save for the following:

|  |  |  |
| --- | --- | --- |
| Environmental Factor | System Influence | Adjustment Factor Value |
| Data Communication | Significant | 4 |
| Performance Objectives | Strong Throughout | 5 |
| Operational Ease | Strong Throughout | 5 |
| Multiple Site Usage | Strong Throughout | 5 |
| Reusability | 50% | 4 |

*Table 4: Non-average environmental factors*

Of the 14 listed environmental factors, the other 9 environmental factors are considered average, thus have an adjustment factor value of 3.

**Step 4 – Calculate Complexity Adjustment Factor (CAF):**

The complexity adjustment factor is simply the total degree of influence of all environmental factors on the function point value. You can get the CAF using the following equation:

Summing each of the environmental factors adjustment factor value gives a total degree of influence of 9\*3 + 4\*2 + 5\*3 = 50. The CAF according to Eq. 1, then is 50\*0.01 + 0.65 = 1.15.

**Step 5 – Compute Adjusted Function Points (FP):**

We now use the complexity adjustment factor to compute the adjusted function points based on the additional complexity environment for this problem. The equation to compute the adjusted function points is as follows:

Given the 748 function points computed in step 2 and the CAF of value 1.15 computed in step 4, we can conclude that the adjusted function points is 748\*1.15 = 860 (after rounding).

**Step 6 – Convert to Lines of Code (LOC):**

According to the problem statement, the implementation for this project will be a Graphic Icon Language. According to the table from lecture, Graphic Icon Languages have an average LOC/AFP of 4. You can compute the number of lines of code for a given language with the following equation

Note that LOC stands for Lines of Code, and AFP stands for Adjusted Function Points, as in previous sections. Based on the AFP from step 5 and the average LOC/AFP of 4 from the lecture table, we conclude that we will need 860\*4 = 3440 lines of code for this project.

**Discussion/Conclusions**

Function points are a unit of measurement which express the amount of business functionality a system/product will provide to an end user. Function points are a unitless measure of software size, and are widely accepted in industry for software sizing and effort estimation. The process for estimation function points can vary from empirical, gut estimates to a more standardized process like the one discussed in the computations section as well as lecture. Function points work off estimating the number of inputs, outputs, inquiries, files and interfaces to give an idea of how large the software project will be.

As previously discussed in the computations section, the proposed project has 860 adjusted function points which translates to 3440 lines of code in a given Graphic Icon Language. This is a relatively small project. Under the assumptions that a programmer can write 10 lines of code per day, and the assumption that the given programmer works 40 hour weeks, the task can be completed in about 8.6 weeks, or 43 working days. For such a small project, I would not recommend having more than one developer on the project unless certain aspects of the project (accelerated schedule, overlapping areas of expertise, etc.) dictate it.