Kaleidoscope Training System

Basis of Estimate

Version 1.0

Release Date: 06 July 2020

CDRL: NA

Prepared for: Kaleidoscope Case Study, JHU 605.60.81.SU20

Authors:

Brian Hoelzer

Emily List

Brian Loughran

Jason Zhao

# Task 1:

This Basis of Estimate was determined using Constructive Cost Model (COCOMO). This model addresses software design, code and unit test, unit and computer software component (CSC) integration into the larger Computer Software Configuration Items (CSCI), CSCI test (or system test) and sell-off. For this project, requirements analysis is a systems engineering task and is therefore not included in this model. For this project, the COCOMO uses an Organic mode and a Detailed model.

Justification for Organic Mode

Organic uses a small team in a familiar environment and with a familiar application. This project contains two subsystems, a PC CSCI and a SUN CSCI running on Windows XP and Unix environments respectfully, and all programming is being done in C++. The expectation is that even junior members of the team would be familiar with this sort of development environment. The project is to create a training system, where groups of students work through units of training material over the course of a week. So, the team is working to a set of less-than-rigid requirements, e.g., data reduction, scientific models, business models, simple inventory control, or simple production control.

Nothing from the SOW indicates that either subsystem would have any embedded mode characteristics. The system seems straight forward, like several other training systems, and does not seem to have complex command and control structures.

Justification for Detailed Model

The Detailed Model adds the phases of the project and subsystems to use the multipliers in the cost driver equations for each phase of project and/or for each subsystem. This model makes the most sense for this project since there are 3 builds, each including the two subsystems. By using the Detailed Model, the team was able to more accurately determine the schedule for each of the builds.

# 

# Task 2:

|  |  |  |
| --- | --- | --- |
| **Attributes** | **Rating/Assumption** | **Justification** |
| RELY | Low | If the system goes down the loss would be, at most, one week of work for the students and instructor. This would be easily recoverable losses. |
| DATA | Nominal | The databases for this training system should be 1-3 MB which would result in a D/P between 10 and 100 |
| CPLX | Nominal | Nominal control operations, low computational operations and device-dependent operations, and very low data management operations. The highest rating is nominal |
| TIME | Nominal | Nothing in the SOW indicates that any of the operations would be particularly time intensive, so absent any other information- assume the average |
| STOR | Nominal | Nothing in the SOW indicates that any of the operations would be particularly storage intensive, so absent any other information- assume the average |
| VIRT | Nominal | It seems that at most there might be a major change to the VM once per build, or about 6 months |
| TURN | Low | Interactive system |
| ACAP | Nominal | Absent any other information- assume the average |
| AEXP | Nominal | Assuming at least 3 years of experience with the application domain |
| PCAP | Nominal | Assuming 55th percentile for average programmers capability |
| VEXP | Nominal | Assuming at least one year of experience working with VM |
| LEXP | High | Assuming at least 3 years of experience with C++. Given that C++ is a common language for degree course work, even new graduates would be familiar with the language |
| MODP | High | General use of modern programming practices |
| TOOL | Nominal | Strong use of small tools and basic use of large tools |
| SCED | Nominal | Assuming familiarity with theyse types of training systems, nothing seems like the system should be particularly difficult or easy so assuming that it should take the average amount of time to deliver |

*Table 1: COCOMO Parameters Justifications*

# Task 3:

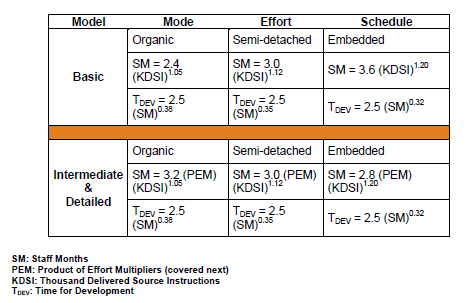
Based on the above assumptions of COCOMO parameters, we can map the effort multiplier ratings into the effort multiplier chart. Each selected value is highlighted and the effort multiplier value is shown in the rightmost column.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Attributes** | **Very Low** | **Low** | **Nominal** | **High** | **Very High** | **Extra High** | **Multiplier** |
| RELY | .75 | .88 | 1.00 | 1.15 | 1.40 |  | .88 |
| DATA |  | 0.94 | 1.00 | 1.08 | 1.16 |  | 1.00 |
| CPLX | .70 | .85 | 1.00 | 1.15 | 1.30 | 1.65 | 1.00 |
| TIME |  |  | 1.00 | 1.11 | 1.30 | 1.66 | 1.00 |
| STOR |  |  | 1.00 | 1.06 | 1.21 | 1.56 | 1.00 |
| VIRT |  | .87 | 1.00 | 1.15 | 1.30 |  | 1.00 |
| TURN |  | .87 | 1.00 | 1.07 | 1.15 |  | .87 |
| ACAP | 1.46 | 1.19 | 1.00 | .86 | .71 |  | 1.00 |
| AEXP | 1.29 | 1.13 | 1.00 | .91 | .82 |  | 1.00 |
| PCAP | 1.42 | 1.17 | 1.00 | .86 | .70 |  | 1.00 |
| VEXP | 1.21 | 1.10 | 1.00 | .90 |  |  | 1.00 |
| LEXP | 1.14 | 1.07 | 1.00 | .95 |  |  | .95 |
| MODP | 1.24 | 1.10 | 1.00 | .91 | .82 |  | .91 |
| TOOL | 1.24 | 1.10 | 1.00 | .91 | .83 |  | 1.00 |
| SCED | 1.23 | 1.08 | 1.00 | 1.04 | 1.10 |  | 1.00 |

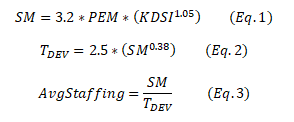
*Table 2: Mapping COCOMO parameters to Effort Multiplier Ratings*

To get the total effort multiplier rating for the project we can simply take the product of each effort multiplier rating. We can use this effort multiplier rating to compute staff months and time to develop for each of the three builds, since the multipliers will likely not change much between the three builds. The result of the product of all of the effort multipliers is 0.66.

As discussed in the above sections, we are using the organic development model and we are using the detailed model for each of the three builds. We can consult the following lookup table to determine the equations to use for the staff months and time for development, as well as the average staffing for the duration of each build.

*  
Table 3: Equation lookup table*

Using the organic model and the detailed equations we see that we can use the following two equations to compute staff months and development time. The equation for computing the average staffing is trivial, but is shown below as well.



The only remaining item to collect is the lines of code to write for each of the builds. We can take each of the tasks provided in the case study and sum the lines of code for each task into a total lines of code for each build. This information was also collected in the SDP. The total lines of code for each build is shown in the table below:

|  |  |
| --- | --- |
| **Build** | **Lines of Code** |
| 1 | 11000 |
| 2 | 13200 |
| 3 | 7800 |

*Table 4: Summary of lines of code to be delivered for each build*

With the above information we can compute the number of staff months, the time to develop, and the average staffing for each of the builds. The results for each of the builds are shown in the below table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Build** | **1** | **2** | **3** |
| SM | 26.27 | 31.81 | 18.31 |
| TDEV | 8.66 | 9.31 | 7.55 |
| Avg. Staff | 3.03 | 3.42 | 2.43 |

*Table 5: Tabulated results for staff months, development time and average staffing*

# Task 4:

The approach taken to compute the effort, schedule and staffing profile for the BOE was to use the COCOMO Model. The process for the COCOMO model was discussed in the first three sections, however a more generalized approach can be written as:

1. Tabulate effort multipliers and produce project effort multiplier

2. Select a development model (organic, semi-detached, and embedded)

3. Choose an equation mode (basic, intermediate, advanced)

4. Collect lines of code estimate for project (or each build if using advanced equation mode)

5. Compute applicable parameters

As a team we met to discuss the input parameters to the COCOMO model (organic/semi-detached/embedded, basic/intermediate/advanced, effort multipliers, etc.). Each of the input parameters was debated and each team member shared their own understanding to get an appropriate value for each of the parameters. Once the parameters were determined, a spreadsheet was created to automatically compute the product of the effort multipliers and the staffing parameters. Finally the team split up to write a human-readable document to summarize the results.

# Task 5:

Overall, the estimations provided by the COCOMO method provide a clearer picture of when we can expect to deliver each build. In addition, the staffing requirements indicates a small team (average size 3) that validates our choice of the organic model.

# Task 6:

In case the software being developed may not be familiar with our team, we can redo the estimates using the semi-detached mode:

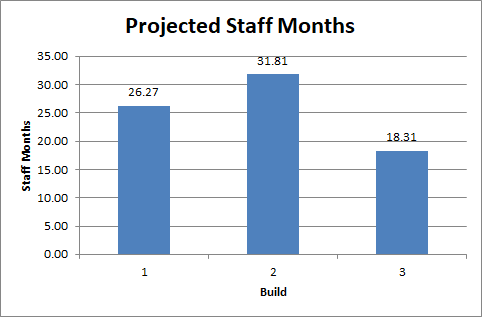
|  |  |  |  |
| --- | --- | --- | --- |
| Build | 1 | 2 | 3 |
| SM | 29.12 | 35.72 | 19.82 |
| Tdev | 8.14 | 8.74 | 7.11 |
| Avg. Staff | 3.58 | 4.09 | 2.79 |

*Table 6: Recalibrated development time estimates*

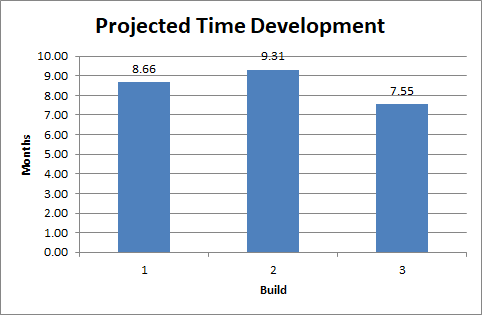
While the estimates for staff months are around 10% higher, it does not substantially alter the timelines for this project.

# Task 7:

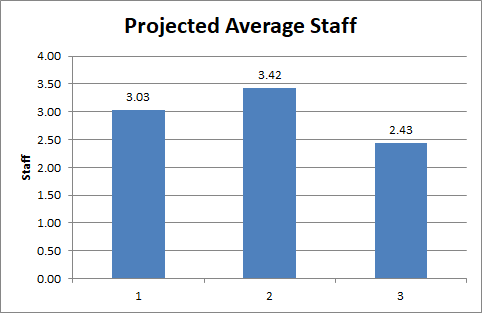
Below are graphs generated from Table N+3 of the predicted staff months, time for development in months, and average staff needed for each build.



*Figure 1: Projected Staff Months*



*Figure 2: Projected Time Development*



*Figure 3: Projected Average Staff*

As per the calculations, for build 1 we are predicted to have approximately three staff members working about 8 and ⅔ months. Build 2 will require approximately 3 to 4 staff members working for 9 ⅓ months and build 3 will require about 2 to 3 staff members working for about 7 and ½ months to complete.

# Task 8:

These staffing requirements and scheduling make sense as given by the number of lines of code for each build (Build 1 requiring 11,000 LOC, Build 2 requiring 13,200 LOC, and Build 3 requiring 7,800 LOC) we would expect the staff requirements and scheduling to reflect this and they do. The staffing for build 2 and development time is the greatest of the three and therefore is predicted to have the most staff and the longest development time. Build 1 is the second longest and this is reflected with the staff and development time being the second longest. Build 3 has the least of amount of LOC and therefore has the least staff and smallest development time.

If we look at the rate of LOC per day predicted by the Basis of Estimate Assignment of 18 LOC per day and perform the following calculations:

11,000 LOC / 18 LOC per day = 611.11 staff days / 5 working days per week = 122.22 staff weeks / 13 weeks per staff quarter = 9.4 staff quarters \* 3 staff months = 28.21 staff months

13,200 LOC / 18 LOC per day = 733.33 staff days / 5 working days per week = 146.67 staff weeks / 13 weeks per staff quarter = 11.28 staff quarters \* 3 staff months = 33.85 staff months

7,800 LOC / 18 LOC per day = 433.33 staff days / 5 working days per week = 86.67 staff weeks / 13 weeks per staff quarter = 6.67 staff quarters \* 3 staff months = 20 staff months

Given these straight linear predictions for how long the development time would take in staff months, we see that they are close to what was predicted in Task 3, taking slightly longer than what was predicted given that we took the conservative number of 18 lines of code.

If we take the more optimistic rate of 20 LOC per day and perform the following calculations:

11,000 LOC / 20 LOC per day = 550 staff days / 5 working days per week = 110 staff weeks / 13 weeks per staff quarter = 8.46 staff quarters \* 3 staff months = 25.38 staff months

13,200 LOC / 20 LOC per day = 660 staff days / 5 working days per week = 132 staff weeks / 13 weeks per staff quarter = 10.15 staff quarters \* 3 staff months = 30.46 staff months

7,800 LOC / 20 LOC per day = 390 staff days / 5 working days per week = 78 staff weeks / 13 weeks per staff quarter = 6 staff quarters \* 3 staff months = 18 staff months

We see these calculated staff months are close to the predicted staff months from task 3 as well, being slightly under. We can be confident that values calculated in Task 3 are fairly accurate.