|  |
| --- |
| Johns Hopkins |
| Module 4 Assessment |
| Development Process and Labor Estimate |
|  |
| **Brian Loughran** |
| **6/20/2020** |

|  |
| --- |
|  |

**Problem Statement**

The website business area at Cyber Statistics Inc. is preparing a proposal for a new contract to develop a robust secure web application using best practices.  The web program must support four releases to be built 12 months apart beginning 14 months after contract award.  The customer requires the product to be fully tested and installed three months before the official releases.  The web program has been estimated to require 60,000 lines of code (12,000 for basic functionality, 12,000 for build 0 unique functionality, 15,000 for build 1 unique functionality, 12,000 for build 2 unique functionality, and 9,000 for build 3 unique functionality).  Your organization has not previously developed in this domain so you are using 10 lines of code per day as a very conservative productivity estimate.

1. What software development process model would you recommend now? State the reasons for your recommendation. If you changed your mind from your original response in Module 2, state why.
2. Does the "quick labor estimate" and "rule of thumb schedule" support your recommendation if the proposal manager wants a very low risk schedule? (Provide answers to two decimal points of accuracy and use 160 hrs/month as average personnel availability.)

**Assumptions**

Before we select a software development process model, we should clarify some characteristics of the project based on the problem statement. While the statement does not specifically lay out things such as whether the architecture has been laid out, whether we would like customer/management visibility, etc. we can still make some inferences based on what is given.

Given the detailed estimates on the number of lines of code, we can make the assumption that the architecture and requirements of the system are well defined. Because of the well defined requirements, there would be minimal need for a development model which allows for midcourse corrections. Because of the detailed schedule and 3 month early deliverables, it may be prudent for the development model selected to have good visibility to both customers and management. There was no mention of integration with other commercial products in the problem statement, thus we can assume that integration with existing systems will at least remain at a minimum. As this is a domain that the organization has not previously developed in, it will require some manager/developer sophistication. And we can always assume that we want to produce a highly reliable system (thus likely will not use something like code and fix).

Some other general assumptions from the problem statement are that overhead such as testing, code reviews, etc. are included in the productivity estimate. Computations are shown to try to determine the number of team members included on the project, and if there is additional time needed for testing and other project activities, then those computations will be skewed. Another assumption is that the number of team members does not change the productivity estimate. In a typical project, as the number of team members increase it becomes more and more difficult to integrate and organize the team. For the given problem, we assume that no matter the size of the team, they can always produce 10 lines of code per day.

**Computations**

**Part A – Quick Labor Estimate:**

The quick labor estimate is a measure of the number of staff months needed to complete a project. With the given staff months we can do computations to determine how many people should be employed on a given project, and raise or lower that number if we want to take more or less risk. The effort in staff months can be computed from the below equation:

The effort computed from this equation is in number of staff months. For example, a value of 10 expected staff months of effort would take a single employee approximately 10 months to complete, or would take 2 employees 5 months.

Doing the computations for the number of staff months for each of the builds then is simple. We use an assumption of 10 lines of code per day of productivity and 160 productive hours per staff month. The result of the computation for the amount of effort for each of the builds is summarized in the table below:

|  |  |  |
| --- | --- | --- |
| Build | # lines to write | Staff Months (SM) |
| 1 | 24000 | 120 |
| 2 | 15000 | 75 |
| 3 | 12000 | 60 |
| 4 | 9000 | 45 |

*Table 1: Summary of Staff Months of effort for each build*

**Part B – Rule of Thumb Schedule**

From the quick labor estimate, we can compute a rule of thumb schedule. The rule of thumb schedule is a widely used formula which also is referred to as the time for development (TDev) or time for labor (TL). Based on the percentile of risk that you want to take on, you can get a lower bound on the amount of time that development will take for a given quick labor estimate. Below are a series of equations used to compute the minimum time for labor based on the number of staff months expected for a project:

Since the project manager wants a low risk schedule, we will use the constant from (Eq. 4). I assume that the lead time for the first build is 11 months (the first release is 14 months after the contract is awarded, however the functionality must be built 3 months before release), and the time for each subsequent build is 12 months (if you start working on subsequent builds during the three month window before release). A software manager should take care that the minimum lead time is less than the scheduled lead time. I also computed the number of team members that should be assigned to the project by simply dividing the number of staff months by the months of lead time. The results for this may be helpful in determining a software development model. The results are tabulated below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Build | # lines to write | Staff Months (SM) | Lead Time (months) | Minimum Lead Time (months) | # team members |
| 1 | 24000 | 120 | 11 | 9.86 | 10.91 |
| 2 | 15000 | 75 | 12 | 8.43 | 6.25 |
| 3 | 12000 | 60 | 12 | 7.83 | 5.00 |
| 4 | 9000 | 45 | 12 | 7.11 | 3.75 |

*Table 2: Summary of Rule of Thumb Schedule*

We can see that for each of the proposed builds, the minimum lead time is less than the actual lead time for the build, even assuming a very low risk schedule (90% completion chance). The additional wiggle room between the minimum lead time and the expected lead time indicates to the software manager that the proposed schedule is very low risk if properly staffed.

Note that the computation on # of team members will give the number of team members needed if each team member works full time on the project. If some team members are working only a fraction of the time, then obviously more team members will be required to complete the project.

**Discussion/Conclusions**

In the assumptions section, we laid out some requirements for the software development process model. The language in the assumptions section was selected to match well with the general capabilities chart for the different software development process models. One assumption that has changed since my initial choice of software development process model in Module 2 is the quality of risk management needed for the software development process model. Given that we should be able to complete the project in the given timeframe with very little risk, we can ignore the previous assumption that the software development model would need to manage risk particularly well. We can use the assertions made in the assumptions section and here to clarify the minimum capabilities of our software development process model. By comparing the actual capabilities of the different development models with the minimum expected capabilities for our project, we should be able to determine which development model is best for the given project. This is done in the below table. For each of the development models, a green box indicates that the lifecycle model meets the minimum requirement of the project, a red box indicates that the lifecycle model does not meet the requirements of the project and choosing that lifecycle model could create challenges down the road, and a yellow box indicates that the lifecycle model does not necessarily meet the requirements for the project, but the severity of the miss is less than that of a red box. One should note that the color of the boxes are somewhat subjective, and there can sometimes be an argument to be made of the color of a specific box, however this type of designation does a good job giving a general idea which lifecycle models to focus on. The box colors, then, follow the general paradigm that green indicates good, red indicates bad, and yellow indicates not ideal. You can see the results below:

*Table 3: Comparing project requirements to different life cycle models*

These results are very similar to those displayed and discussed in detail in the Module 2 deliverable, thus for brevity I will summarize the discussion from Module 2.

Code and fix is largely outdated, does not produce reliable systems, and does not manage risk, thus was not chosen. Waterfall does a poor job with providing customer visibility, thus was not chosen. COTS was not chosen due to no indication in the problem statement that the customer is looking for a COTS solution. Agile was not chosen due to its poor ability to be constrained to the rigid predefined schedule described in the problem statement.

The two final models (spiral and evolutionary) are both likely good choices for the given project, and either would likely work well for the project in the problem statement. However, you cannot develop software using two development models at once, so we must choose between the two. I give a slight edge to evolutionary over spiral, the same model chosen from Module 2.

The spiral model would likely be a good choice for our project; however it is not the best choice. Spiral typically does a great job on high risk projects and eliminating errors early. Due to the incremental nature of the releases, this would be important to the project. Spiral also does a good job managing long term projects, providing visibility to stakeholders, and managing teams of this size, which are also important to this project. However, typically the spiral model produces a prototype for each of the development iterations, rather than a true deliverable which the customer is expecting. The spiral model also often incurs more cost than other models, causing it to be not widely used as some of the more established models. While spiral would likely work well with the given project, the described weaknesses gives a slight edge to our final choice of development model.

The evolutionary model is the best model for the given project specifications. The evolutionary model is used when you want to deliver subsets of the functionality periodically, as well as when the interface/domain is not well defined, which is the case for the given project. Each of the builds in the developmental model goes through a complete cycle of tailored activities, which helps to ensure the quality of each build. The evolutionary model also does a nice job of managing long term projects, providing visibility to stakeholders, and managing teams of this size which are other important components of the development life cycle for this project. Because of how well the evolutionary model is tailored to the specified project, it makes sense to use the evolutionary model in this case.