JOHNS HOPKINS

Module 2 Assignment

Recommending a Development Process

Brian Loughran

6/7/2020

Brian Loughran
Software Project Management
Johns Hopkins
Module 02 Assignment: Recommending a Development Process

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.

What software development process model do you suggest? Defend your suggestion.

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. Based on the detailed schedule given, we can also assume that we would want a development model which both manages risks well and can be constrained to a predefined schedule to avoid missing deadlines. Building off of this, 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.

Brian Loughran
Software Project Management
Johns Hopkins
Module 02 Assignment: Recommending a Development Process

Computations

Using 10 lines of code per day as a conservative estimate, we can compute the number of person-hours for each of the builds that are needed to meet the requirements of the project. We can do this with a simple division:

$$personhours = \frac{\#\ lines\ to\ write}{productivity\ estimate}$$

(Eq. 1 – computing person-hours)

Person-hours is a measure of total effort. A task that is estimated to take 80 person-hours would take 1 person working two 40 hour weeks, or two people working one 40 hour week each. From the problem statement, we know that we are estimating 10 lines of code per day, thus that will be our productivity estimate. Dividing the number of lines of code by our rate will give us the amount of effort needed to complete the requirements of each of the builds.

Further, we can determine the minimum number of people we will need on our development team to meet the requirements for each of the builds. We can do this by dividing the amount of time for each of the builds by the number of person-hours that we got from Eq. 1. Thus, we have the following equation:

$$\#\ team\ members = \frac{personhours}{build\ time}$$

(eq. 2 – computing minimum number of team members)

Note that this 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. Also of note is the lack of units in both Eq. 1 and Eq. 2. These are generalized equations, and the units will be taken care of in the actual computation.

Thus, for each build we can compute the minimum number of team members required. For this exercise, I assume 40 hour weeks, and exactly 4 weeks in a given month. I also 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). The results are tabulated below:

# lines to write	lead time (months)	person-hours (hours)	# team members
24000	11	19200	10.90909
15000	12	12000	6.25
12000	12	9600	5
9000	12	7200	3.75

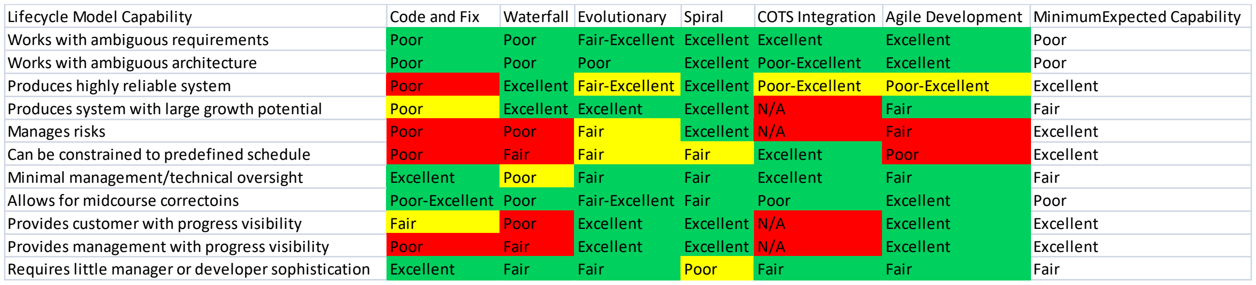
Brian Loughran
Software Project Management
Johns Hopkins
Module 02 Assignment: Recommending a Development Process

Table 1: Estimation of project team size

After this we start to get a better idea of what our project team will look like. For the duration of the project, we will need between 4 and 11 team members working full time. This gives us some perspective for determining what software development process model we will employ.

Discussion/Conclussions

In the assumptions section, we laid out some requirement for the software development process model that we will eventually employ. The language in this section was selected to match well with the general capabilities chart for the different software development process models. We can use the assertions made in the assumptions section 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:



Lifecycle Model Capability	Code and Fix	Waterfall	Evolutionary	Spiral	COTS Integration	Agile Development	MinimumExpected Capability
Works with ambiguous requirements	🟩 Poor	🟩 Poor	🟩 Fair-Excellent	🟩 Excellent	🟩 Excellent	🟩 Excellent	Poor
Works with ambiguous architecture	🟩 Poor	🟩 Poor	🟩 Poor	🟩 Excellent	🟩 Poor-Excellent	🟩 Excellent	Poor
Produces highly reliable system	🟥 Poor	🟩 Excellent	🟨 Fair-Excellent	🟩 Excellent	🟨 Poor-Excellent	🟨 Poor-Excellent	Excellent
Produces system with large growth potential	🟨 Poor	🟩 Excellent	🟩 Excellent	🟩 Excellent	🟥 N/A	🟩 Fair	Fair
Manages risks	🟥 Poor	🟥 Poor	🟨 Fair	🟩 Excellent	🟥 N/A	🟥 Fair	Excellent
Can be constrained to predefined schedule	🟥 Poor	🟥 Fair	🟨 Fair	🟨 Fair	🟩 Excellent	🟥 Poor	Excellent
Minimal management/technical oversight	🟩 Excellent	🟨 Poor	🟩 Fair	🟩 Fair	🟩 Excellent	🟩 Fair	Fair
Allows for midcourse correctoins	🟩 Poor-Excellent	🟩 Poor	🟩 Fair-Excellent	🟩 Fair	🟩 Poor	🟩 Excellent	Poor
Provides customer with progress visibility	🟨 Fair	🟥 Poor	🟩 Excellent	🟩 Excellent	🟥 N/A	🟩 Excellent	Excellent
Provides management with progress visibility	🟥 Poor	🟥 Fair	🟩 Excellent	🟩 Excellent	🟥 N/A	🟩 Excellent	Excellent
Requires little manager or developer sophistication	🟩 Excellent	🟩 Fair	🟩 Fair	🟨 Poor	🟩 Fair	🟩 Fair	Fair

Table 2: Comparing project requirements to different life cycle models

Code and fix does predictably poor when comparing to the capability needed for this project. Code and fix is largely outdated, and would not produce the reliable system needed, nor does it properly manage risk, conform to schedule, or provide visibility to stakeholders. Further, code and fix is typically only suitable for projects with only one developer, and we have already discussed that we need a team of 4-10 people throughout the duration of the project.

The waterfall method also seems to be a poor choice for the minimum expected capability for this project. While this project seems well-defined, and the waterfall method typically does great with well-defined projects, the waterfall method does a poor job managing risk, conforming to schedule, and providing visibility to management and customer stakeholders. Because of this, we will not select the waterfall model for this project.

Next we can analyze the COTS integration model. While COTS is typically beneficial in terms of reducing development time and the ability to produce highly reliable products because of the field tested nature of the COTS products, there is no indication in the problem statement that the customer is looking for a COTS solution. Because of this, it does not seem prudent to select the COTS integration model.

Agile is a widely used framework which does a great job handling changing requirements and changing market strategies. But based on the problem statement, we made the assumption that all the requirements of the system are already well-defined, which would negate the greatest strength of the agile methodology. For that reason alone it may not be prudent to select agile. Coupled with the fact that the agile methodology does not manage risk nor conform to predetermined schedule to the extent that we probably want for this project, it does not make sense to select the agile methodology for this project.

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, and I …

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 risk, 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. Furthermore, the spiral model typically is used for projects with a duration of 5+ years, and while the 4 year development cycle for this project is of similar magnitude, the project may not be best suited to the spiral model. 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 risk, 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.