University of Ontario Institute of Technology (UOIT)
Faculty of Engineering and Applied Science (FEAS)
Department Of Electrical, Computer, And Software Engineering (ECSE)

Software Project Management (SOFE 3490)

Lab #3 Submission | COCOMO and Activity Diagram

Okwudili Ezeme

Winter 2020

Group Members
Aryan Kukreja (100651838)
Emil Ilnicki (100659072)
Jamieson Leibovitch (100612845)

Project Estimation

We chose to construct a class application that allows students to remain in touch with their classmates and alumni after graduation. Using COCOMO we have a choice between three types of systems.

1. Organic
2. Semi-detached
3. Embedded

The class application will fall under a semi-detached system as we haven't encountered this problem before, we are using various platforms and our team size is relatively small. In addition, our class application will have approximately 15,000 LOC. Using COCOMOs estimation of effort equation our $a = 3.0$ and $b = 1.12$ thus giving us an effort value of

$$
\begin{aligned}
E &= a(KLOC)^b \\
&= 3.0(15)^{1.12} \\
&= 62.28 \text{ person-months}
\end{aligned}
$$

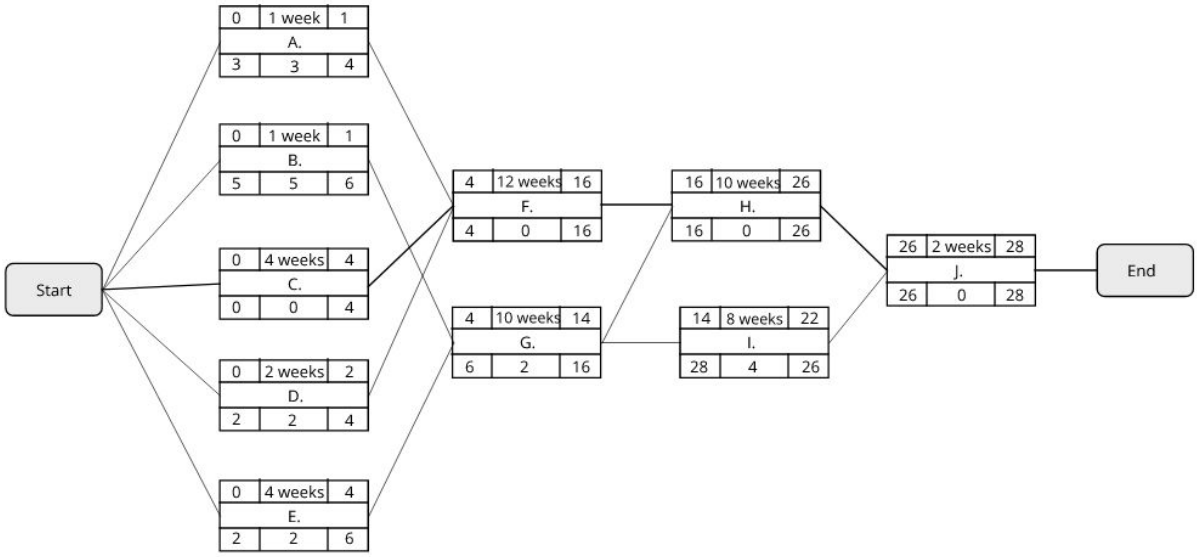
Using Albrecht complexity multipliers for function points we will be looking at logical interface files (LIF), external interface file types (EIF), external input (EI), external output (EO) and finally external inquiries (EQ). With these we can estimate the total number of function points that this application will require.

External User Types	Estimate	Complexity	Total
EI	6	4 (Medium)	24
EO	10	5 (Medium)	50
EQ	4	3 (Low)	12
LIF	4	10 (Medium)	40
EIF	2	10 (High)	20
		Total Function Points (FP)	146

Activity Diagram

Nodes:

- A: Development Server Setup (1 wk)
- B: Mobile Env Setup (1wk)
- C: Data collection (4wks)
- D: Create software architecture for server (2wks)
- E: Design UI/UX (3wks)
- F: Server Development (6 sprints, 12wks)
- G: Mobile application Development (5 sprints, 10wks)
- H: Quality Assurance (5 sprints, 10wks)
- I: User Acceptance Testing (2 sprints, 2wks)
- J: Set up production server and deployment (1 sprint 2wk)



Risk Analysis

Running a risk analysis of this project will ensure that contingencies are met in advance prior to the initiation of the project. This, in turn, will lead to reduced overhead costs in the project, because checking risks in advance will lead to fewer errors during the project, causing less time spent in the reworking of components, re-planning, etc…

The risks that have been identified as part of the project are identified and detailed below for further reference:

Technology Setup Compilations

One of the main risks of this project comes in the setup phase itself. In this phase, the staging environments and containers will be created and prepared for use as a base for further development of the project. This poses a risk; by fully defining pipelines, containers, and staging environments, the development team will be restricting itself to working only with those technologies that are compatible with the pre-established pipelines and containers.

In the future, should there be a need to use or integrate technology or programming language that is non-native to these pipelines or containers, then a lot of work will have to be undone to re-adjust or replace these containers/pipelines with those compatible with the desired technology.

To work around this, a lot of prior research, study, and analysis will have to be conducted into the project to understand its technical needs and ensure that right containers, staging environment and pipelines are chosen to satisfy these needs.

Unclear Requirements

The requirements for this project are very short and broad. Many of the requirements can be misinterpreted when converted into specifications for the developers, and this could lead to an error in the design of the final product.

To overcome this shortfall, the team has chosen to follow the Agile software development process. With this, the software will be developed in increments (or in sprints if the Scrum process is specifically undertaken), tested and delivered to the client. The client can then identify any problems he/she has with the delivered increment, allowing the developers to rectify it in advance and prevent it from

happening in subsequent increments. This will help the developer clarify the requirements with the client while keeping to delivery timelines.

Missing Sprint Deadlines

Since each increment is delivered to the client, missing a sprint deadline can lead to a delay in delivering an increment to the client. This will have a domino effect: all subsequent increments will be delayed as a result.

The best way to avoid this risk will be to ensure that increments are sized manageably; the developers will split the project's specifications into increments so that they can meet the deadline; the approach to use here is **under-estimating**: it would be better to expect less from an increment, and actually get more done, rather than over-expect from a delivery and cause a delay.

Unclear Client Design

This risk is similar to unclear requirements: given that the client design is majorly broad, there is a lot of room for misinterpretation.

To overcome this, the developers will need to interface with the clients often: prior to the development of critical increments, a design model of the increment can be discussed with the client prior to starting implementation. This process can be made easier if the client sends a software-quality assurance (SQA) member/group to integrate within the development team who understands the requirements of the project so that he/she can guide the development process to follow the desired requirements. This SQA sub-team will continuously audit design models and implement increments for conformance to the client's expectations.

Inconsistent Data

This refers to the data samples that the client provides to test the increments or to be used in the development process itself. Inconsistencies in these can lead to conflicting, missing or vague requirements.

To overcome this, the team will have to spend a bit of time cleaning the provided data to ensure that problems in the requirements do not arise in the future, if they are addressed with the client now. This will have to be followed every time the client provides some new data, or the developers request it.