**Process Management Report**

February 20, 2017

Team MangoWEB

Caleb Warren

Collin Epstein

Julian Bertini

**Introduction:**

This report establishes the Process Management Plan that will ensure Team Mango WEB’s development of a videogame remains on schedule, within a reasonable cost, and meets its standards for quality. We will outline the expected effort as calculated by the COCOMO II model and using the Top-Down estimation approach. Given the effort calculated by COCOMO II, we will accordingly draw out a high-level schedule plan for the project. In addition, we will describe our team’s quality plan, risk management plan, monitoring plan, and a more detailed schedule.

**1.** **Expected Level of Effort in Person-Months**

Calculation for the expected amount of software in LOC:

Game controller script that will control the state of the game: 300 LOC

Character controller script: 200 LOC

Object behavior (blocks, power-ups, villains): 500 LOC

Puzzle manager and generator: 500 LOC

Environment generator: 400 LOC

Camera controller: 100 LOC

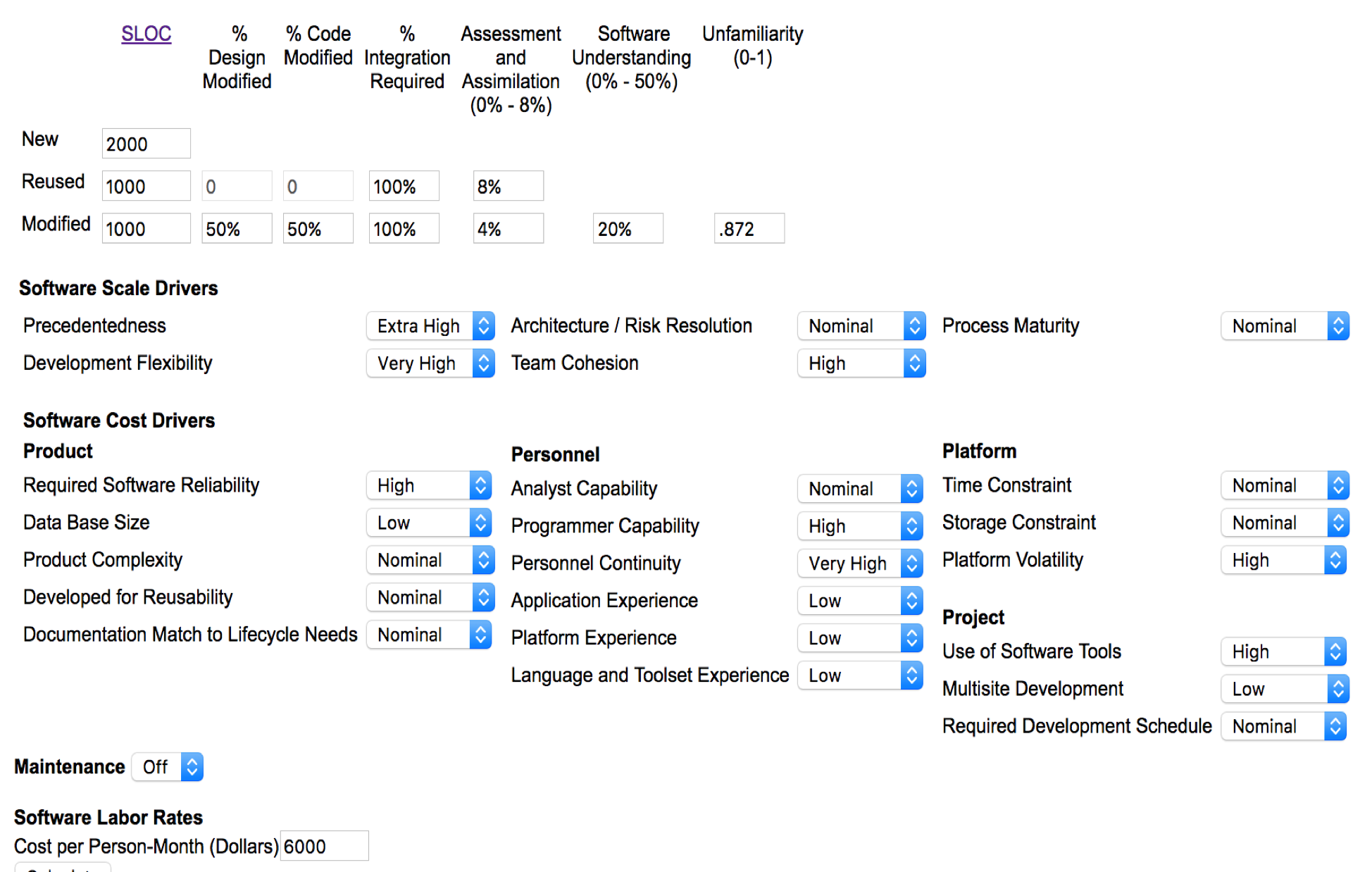
Total LOC: 2000

Since the Unity game engine provides significant built-in functionality, our code will simply add more specific behavior for the functionality that Unity already provides. We will primarily script game and object behavior and allow Unity to implement 3D graphics and baseline program architecture.

The tools provided by Unity obscure its source code, making it difficult to know how much code we will be reusing and/or modifying. Since graphics make up a significant proportion of any videogame project, we assume that about half the lines of code used in our project will be reused or modified code found in the Unity Asset Store or the Unity game engine itself.

We proceed to calculate our expected effort using COCOMO II:

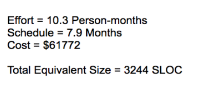
Inputs:



* Precedentedness - programming a video game is a project with precedent.
* Development Flexibility - small, collocated team.
* Architecture/Risk Resolution - carefully constructed risk management plan.
* Team Cohesion - small team, good communicated, aligned goals.
* Process Maturity - Iterative Process Model is well-established.
* Required Software Reliability - game must be playable.
* Database Size - we don’t anticipate using any database at all.
* Product Complexity - we make no assumptions about product complexity
* Developed For Reusability - will develop a game with reasonable replay value.
* Documentation Match to Lifecycle Needs - given our small team size, we hope to minimize documentation.
* Analyst Capability - our team has faith in our general analytical capabilities.
* Programmer Capability- our team has experienced programmers.
* Personnel Continuity - we expect no personnel turnover.
* Application Experience - we have little experience in developing video games.
* Platform Experience - we have little experience developing mobile applications.
* Language and Toolset Experience - limited experience with C# and Unity.
* Time Constraint - project planned to fit within the specified time constraints.
* Storage Constraint - anticipate fitting typical size bounds of a mobile application.
* Platform Volatility - must ensure compatibility with a wide array of mobile devices.
* Use of Software Tools - Unity will handle graphics and high-level architecture.
* Multisite Development - our team is colocated.
* Required Development Schedule - specified a reasonable project for time frame.

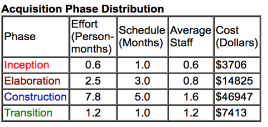
We assume a Software Labor Rate of $6,000 per person-month ($72,000 per person-year), as specified by our manager (Dr. Locke).

COCOMO II Effort Calculation Results:



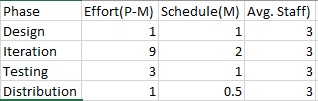
**2.** **Overall High-Level Schedule for the Project**

COCOMO II Schedule Suggestion:



COCOMO II calculates the schedule using RUP (rational unified process), however we are using the Iterative Process Model. We suspect that our schedule will be very similar to this schedule. The ‘design, implementation, and requirements’ categories will most likely have an increase in Person-Months, but this change should be negligible for all intensive purposes. Using the effort calculated by COCOMO II, we built our own high-level schedule. M≈√E, so M≈ 3.1.

Our High-Level Schedule:



**3.** **Quality Plan**

The objective of our quality plan is to reduce the number of defects introduced and increase the number of defects removed from our game. We will achieve this goal by establishing acceptance criterion for the code that we merge into the development project and by setting criterion for how many defects we find acceptable for each deployment.

We plan to do a removal of defects after each phase of each iteration cycle. Namely, we plan to make sure that our acceptance criterion are met after the requirements analysis, design, and coding phases. We will perform unit testing on each feature that is added to the project, and only after it passes our tests will the feature be reviewed by a teammate that is not the person who coded the feature. If it passes that review, then it may be merged into our development project. What’s more, we will do integration testing at the end of each iteration cycle to make sure that the new features are fully compatible with the existing project. We will use Github for code reviews and version quality control to ensure that all members of the team have the most up-to-date version of the project.

In order to decrease the number of defects introduced, we will set stylistic standards for coding. We plan to use Object-Oriented programming and practice writing legible code. More specifically, we want to use well-documented and conventional code that makes proper use of whitespace.

**4.** **Risk Management Plan**

We foresee four major risks for our project:

1. User interface of game may be confusing or not intuitive
2. Graphics Integration
3. Team member becomes unavailable for some reason
4. Developing unnecessary features

The user interface being unintuitive poses the highest risk (medium probability) because usability is usually most difficult to overcome for programmers. It would also have a high cost as a non-usable game is a useless game. This results in a high Risk Exposure. Mitigation will go as follows: diligent testing of the final product will occur, so that a user interface will be optimized as far as we know.

Not developing our own graphics for this game comes with its own risk: integration of graphics to our functionality. The odds of this happening should be low, as Unity is usually programmer-friendly when integrating graphics packages with written code. The cost would be high because no graphics yields an unusable game. We believe this results in a medium Risk Exposure. Well documented code, good understanding of how Unity manages the construction and redirection of its graphics, and good names for functions in our programs will help mitigate this issue.

If a team member becomes unavailable for an extended period of time, our project progress will be hindered significantly. Although the odds of this happening is low, the cost of a lost team member is high, resulting in a low Risk Exposure. There isn’t much we can do for mitigation besides practice good health measures.

Finally, the development of unnecessary features has a medium risk, due to our inexperience in game design. However, we anticipate the cost would be low, since additional features still would likely add to the gameplay experience. We believe this results in a low Risk Exposure. This risk will be mitigated by carefully identifying and prioritizing essential game requirements and features.

**5.** **Project Monitoring Plan**

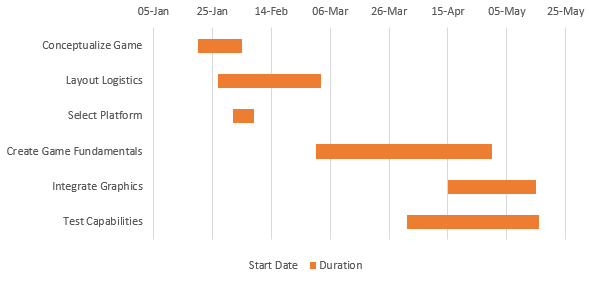
To monitor effort, each team member will track the amount of time they spend working on the project and how many LOC they write. More specifically, time will be logged in association with some specific feature that is currently being worked on. This will allow us to measure effort in relation to each phase of development and particular feature of the project.

The individual progress data may then be used to gauge overall progress by comparing it to the estimated effort generated by COCOMO. This will give us an estimate of whether the project is on schedule.

In addition, we will track the number of defects found along with those that are resolved. We will use some online software such as JIRA to track defects and assign them for development.

**6.** **Detailed Scheduling**

Gantt Chart:



The Gantt Chart above describes our current detailed schedule. We plan to update the detailed schedule at the conclusion of every iteration cycle. Our schedule includes work we have already done and currently specifies a plan for us to finish this project on time.