Project Evaluation

For Multiagent Control of Traffic Signals

Version 1.0

Submitted in partial fulfillment of the requirements of the degree of MSE

Bryan Nehl
CIS 895 – MSE Project
Kansas State University

Table of Contents

1 Introduction

In this document I will evaluate and discuss my views on: the overall project process, the products that were produced, the quality of the products and finally potential future expansion of the project.

1.1 References

Nehl, B. (2012). Software Quality Assurance Plan version 1.0.

Nehl, B. (2012). Project Plan version 2.0.

Fiore, N. (2007). The Now Habit: A Strategic Program for Overcoming Procrastination and Enjoying Guilt-Free Play. CA: Tarcher.

Henny, K. (2010). 97 Things Every Programmer Should Know. Sebastopol, CA: O'Reilly Media.

2 Process

2.1 Problems Encountered

I initially experienced a roadblock with creating the road network structure I needed for my simulation. I eventually overcame the problem, by breaking it down to creating a simple T network based on a four way cross intersection. From there, I was able to build out my network. I had spent a lot of time trying to find a tool that could be used for creating the network. I should have time boxed that research initially and then jumped into the manual approach.

I spent what seemed to me to be a lot of time with the formal specification. Late in the process I discovered that I had older version 2.5 (June 2009) of USE and the current version is 3.0.0 (Sep 2011). That could have made some difference. While working with the USE/OCL tool I feel like I lost focus on the problem/solution I was trying to specify because of focus on how to express my intent in OCL. Also, USE seems more geared towards the specification of object oriented problem-solutions and my project has a strong service oriented distributed asynchronous architecture. Perhaps some other formal tool would have been better suited? In the future I might consider a different tool for the formal specification. I would also time box that research and seek outside help/guidance from an expert.

I opted to try and use "cloud" resources for some aspects of the project. This caused problems when an internet connection was not available. I established a "Plan B" which was to make the notes I would normally make in the web based application in my local OneNote notebook. Later when connectivity was restored, I would updated the web application from my notes.

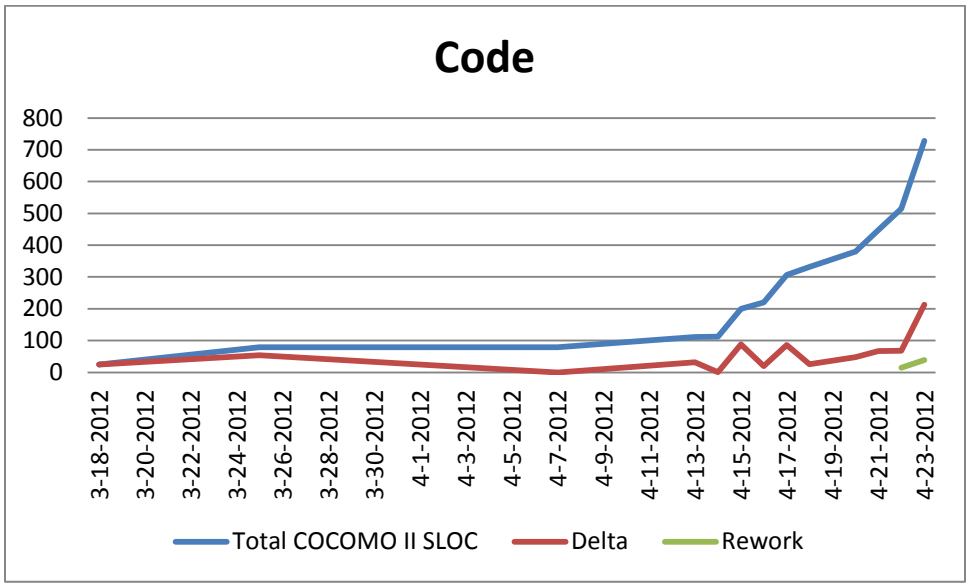
I was also a bit apprehensive about including some references in my public git repository because of copyright concerns.

2.2 Code

This section takes a look at the quantity of code produced, how that corresponds with the project estimates and the quality of the code produced.

2.2.1 Quantity: Source Lines of Code

During the coding portion of Phase III I used PyMetrics to capture the COCOMO II Source Lines Of Code (SLOC) metric. One aspect that was interesting to me was when I was able to add functionality with a minimal movement in the SLOC metric because I was refactoring and reusing code.



2.2.2 Estimation: Comparison between projections and actual

During the creation of the Project Plan I used some early design estimation tools and arrived at an estimate of 6400 SLOC. My own estimate based on some experience with initial spike work was ~1500-2000 SLOC. The actual SLOC code produced was *728*, calculated by the PyMetrics tool.

2.2.3 Quality: Rework

In the SQAP document I stated that I would use the rework metric as a measure of code quality. The metric as presented in [7] is:

$$RW = \frac{E_{Rework}}{E_{Development}}$$

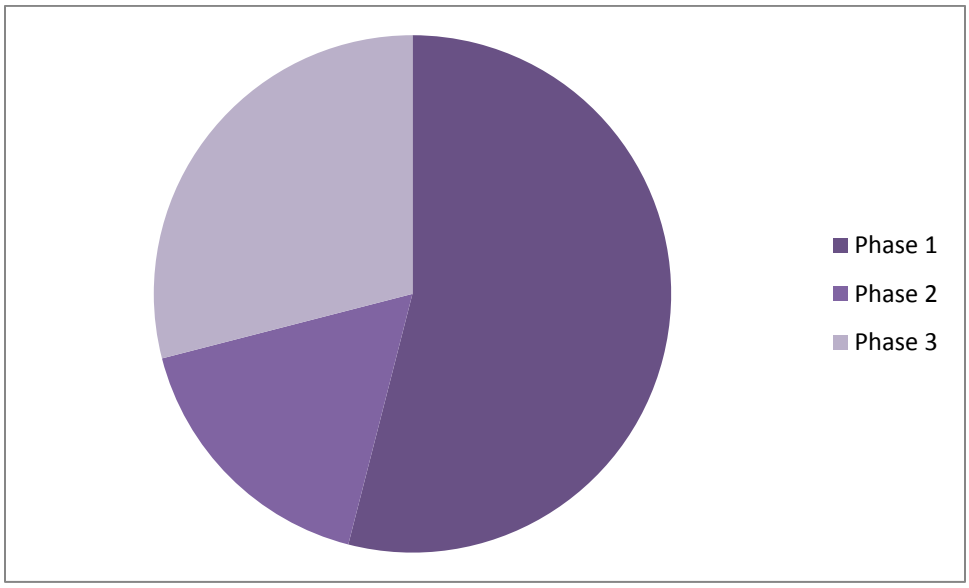
E_{Rework} is the amount of effort spent in rework. $E_{Development}$ is the overall amount of development effort. I will be able to provide this metric by reviewing my engineering notebook and viewing effort as a unit of time. Rework will then be a decimal between 0 and 1. For this project, my goal is to keep the rework below 0.20.

There is hidden development time in that I was able to reuse some code or approaches from earlier spike sessions.

My RW = (112/2673) = 0.0419

2.3 Project Duration

Minutes spent in each phase:
Phase 1: 11704
Phase 2: 3684
Phase 3: 6290+



Only about 12% of the project time was spent coding.

2.4 Lessons Learned

Doing a formal specification did help me think about how the safety operations were going to work.

Doing upfront spikes and primers in the risk areas of SUMO, TraCI, Python, RabbitMQ and git was beneficial when it came to doing the implementation.

While working on this project I read the "Now Habit". One of the big things I got out of this book is to focus on starting. Don't get so overwhelmed with all that there is to do and become paralyzed by it. Ask, "When can I start working on the next task and what is the next task?"

Having done a vision document with System Requirements was beneficial in that it contained the answer to, "what task is next?" Which also tied into the idea of "know your next commit" that I picked up from "97 things" book.

Multiple git repos worked well. In the future I would include the OneNote notebook in repository. I would also likely have a secondary repository with tools.

Another idea I picked up from "97 things" was to include the units of measure in my variable names for clarity. For instance the variable named total_CO2_mg refers to the total amount of Carbon Dioxide (CO_2) produced in *mg* (milligrams).

Being able to refer to a design was helpful.

While I had a Gantt chart, I really did more flow based task management. Further into the project the cone of uncertainty narrows. Estimates can refine/narrow as progress is made.

Having a secondary machine was useful for spiking/testing alternate servers, etc.
In the future I would consider using a virtualized environment for testing changes. A Fast hard disk or SSD would have been beneficial.

Having trade ebooks on related technology (RabbitMQ, Python, MongoDB) was beneficial when I couldn't find what I was searching for on the web, it was in a book.

Including SR #'s in code for traceability.

3 Products

3.1 Quality

3.1.1 Rework Ratio
From the rework ratio computed above 0.0419 is considerably better than the .20 goal.

3.1.2 Test Coverage
The Critical Stop Light Code was 100% tested code.

3.1.2.1 Code Standard (PEP8)
The PEP8 code standard was verified at check ins.

3.2 Future Work
This section covers possible future work.

3.2.1 More Generic
I believe that some of the code could be made more generic. For instance the reactive agent code could be further extracted down to a base class.

3.2.2 Configuration File
A configuration file could be made for agent RabbitMQ credentials.

3.2.2.1 Between Agents
The agents could be extended for cooperative action.

3.2.3 Efficiency
It may be possible to use SUMO/TraCI subscriptions for pulling the metrics more efficiently.

3.2.4 Security
More exploration/understanding of RabbitMQ permissions

3.2.5 Teardown
Investigate why the Metrics Agent and Communications Agent do not cleanly tear down.