Project Evaluation

For GMoDS Visualizer and Test Driver

Version 1.0

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

Mike Fraka
CIS 895 – MSE Project
Kansas State University

Table of Contents

1 Introduction

This document is the evaluation of the GMoDS Test Driver and Visualizer project and product.

1.1 References

1. W. Royce, Software Project Management: A Unified Framework, Addison-Wesley, 1998, p. 198.

2 Process

2.1 Problems Encountered

This section lists problems encountered on the project.

2.1.1 Drawing with Java 2D

I had little experience drawing with Java 2D prior to the project. It so happened that in parallel with this project I was assigned a project at work that also required I use this capability. The two efforts were mutually beneficial.

2.1.2 Scrolling a Zoomed Drawing

I had no appreciation for the requirements of placing a customized Component that changes size in a JScrollPane and no experience with zoom on an image.

2.1.3 Designing an Indirect Routing Algorithm for Relations

The projects' top technical challenge was the algorithm for routing indirect relations avoiding intersections with goals and minimizing crossing of relations.

2.1.4 Supporting the Modification Parameter Origin Value

GMoDS does not directly support the "Modification" parameter origin value. My simple solution was to revise a copy of InstanceParameters (called ModifiableInstanceParameters) to persistently track the value of each parameter. Any time a parameter value changes the "Modification" parameter value origin could be attributed to it.

GMoDS has an unresolved issue regarding the behavior of the InstanceTree method modifyInstanceGoal with respect to generating a proper parameter origin value for child goals when their parent is directly modified. In my opinion, the parameter origin value displayed for these child goals parameters that match the modified parameter in the parent should be "Inherited" (I). However, GMoDS sends the value of the "inherited" property for these parameters as false when it calls the ChangeManager method notifyInstanceGoalModified for the child goals. If GMoDS sent the value of "inherited" as true, the GMoDS Visualizer would display a parameter value origin of "I" instead of "M" (modified).

2.2 Estimates

This section compares cost estimates for source lines of code and project duration estimated in phases 1 and 2 to the final totals.

2.2.1 Source Lines of Code

Table 1 Estimates of SLOC by phase

Phase	SLOC Estimate
Phase 1	4K
Phase 2	7K
Phase 3	8.3K (Actual)

Table 1 above shows the estimated number of source lines of code by phase of this project. This table shows that the estimate improved with feedback through the process and came within 1300 lines of the actual value.

2.2.2 Project Duration

Table 2 Comparing estimated and actual durations

Phase	Estimated End Date	Actual End Date
Phase 1	11/10/2010	11/22/2010
Phase 2	2/14/2011	1/13/2011
Phase 3	4/14/2011	3/16/2011

Table 2 above shows that I underestimated the time required for phase 1 but overestimated the time for the final 2 phases. Figure 2 below shows the percentage of time spent in each phase. Phase 1 took the most time as understanding of the project and Java 2D was a time intensive process. Phases 1 and 2 were less demanding.

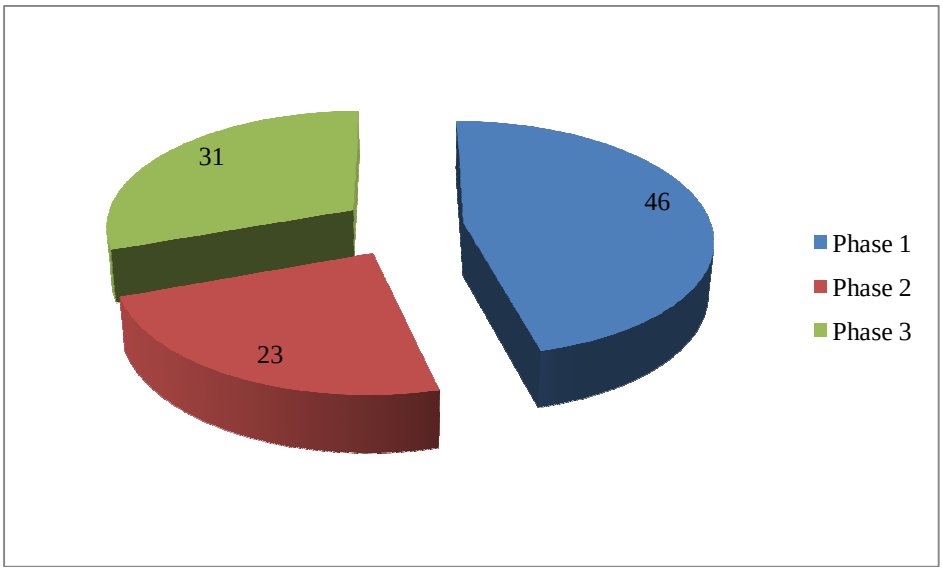

Figure 1 Percent of time spent in each phase

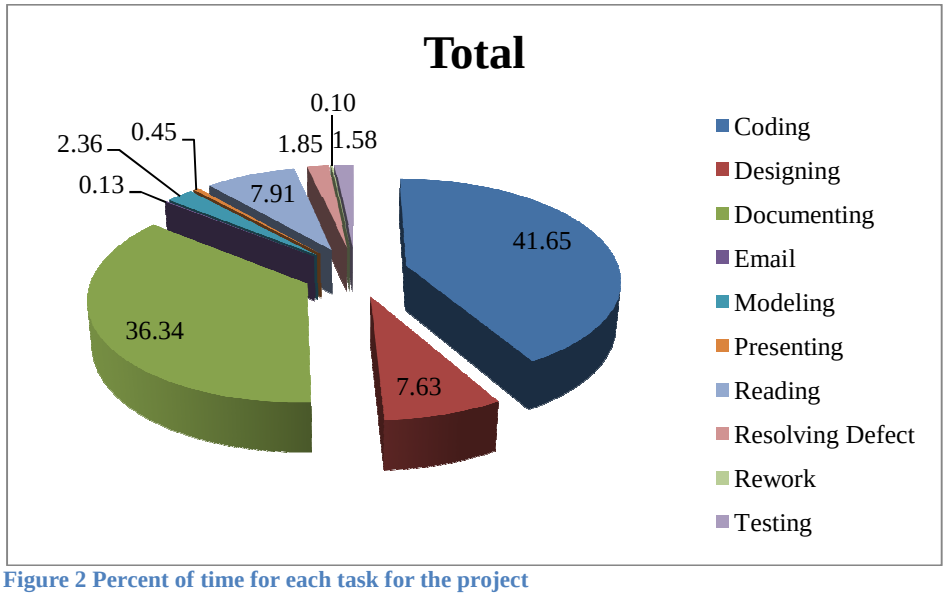

Figure 2 Percent of time for each task for the project

Figure 2 above shows the percent of time spent on each type of task for the entire project. Figure 3 below shows the same information for just phase 1.

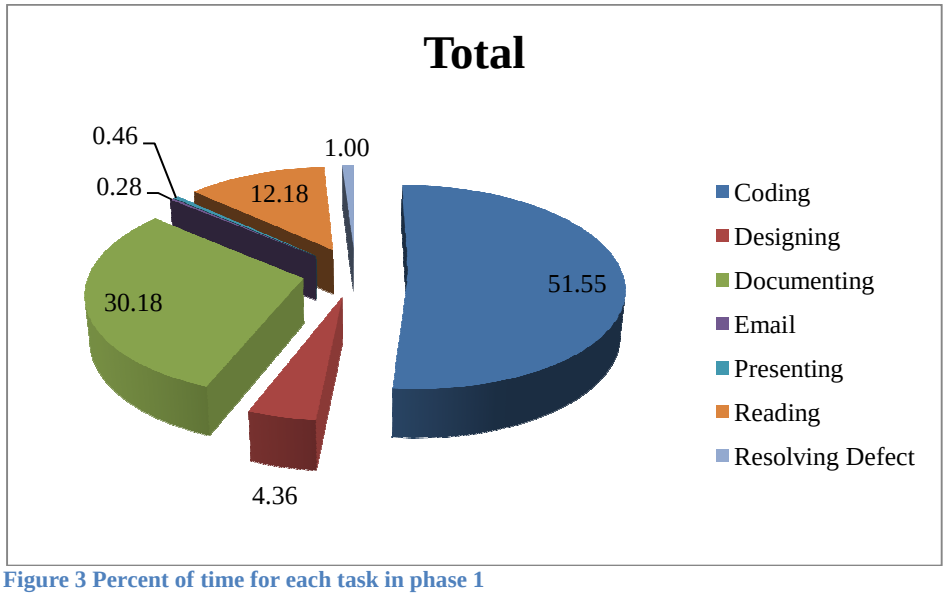

Figure 3 Percent of time for each task in phase 1

Figure 4 and Figure 5 below show the same break down for phases 2 and 3, respectively.

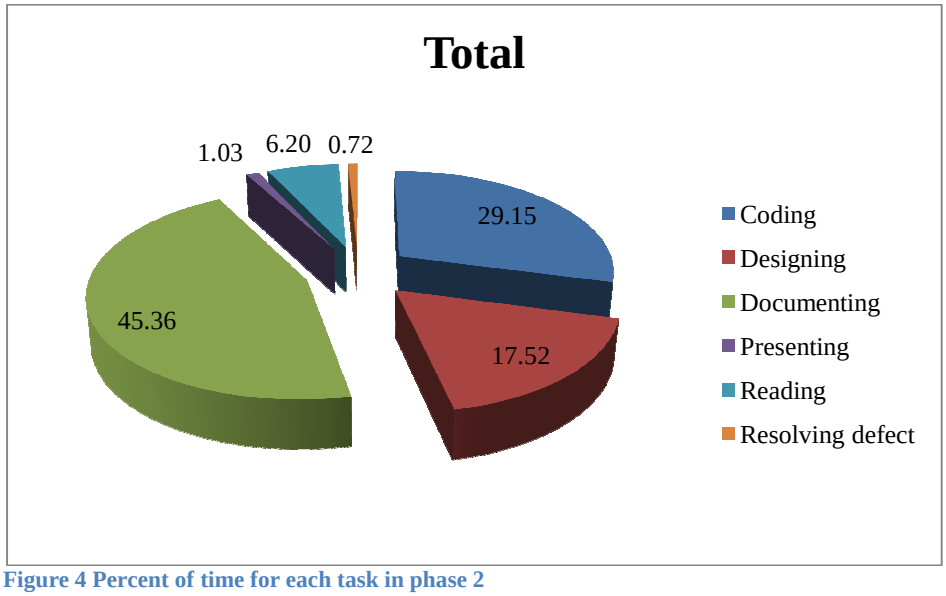

Figure 4 Percent of time for each task in phase 2

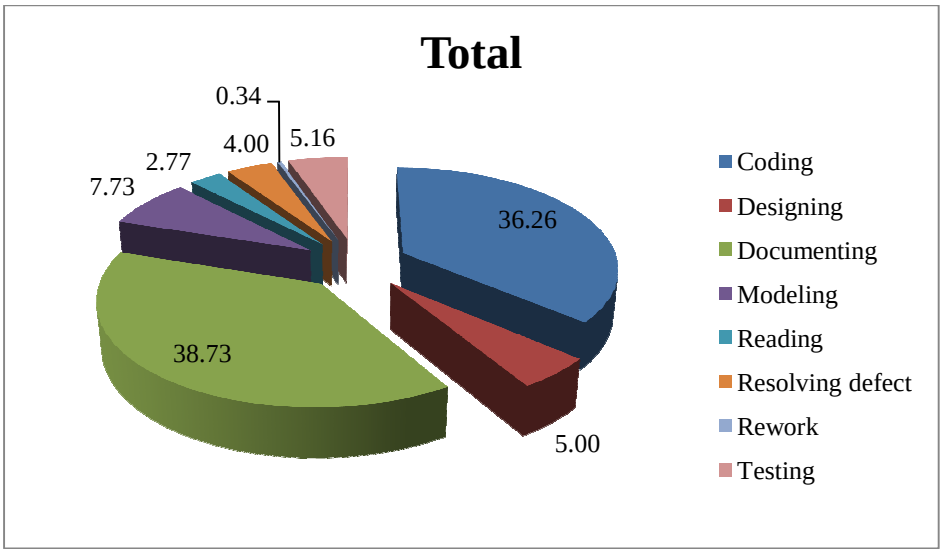

Figure 5 Percent of time for each task in phase 3

Table 3 below shows the actual time in hours spent in each task by phase. Phase 1 was most intensive for coding and reading due to the time needed to understand the project requirements and the Java 2D technology. Testing occurred in phase 3, as planned, leading to most of the rework occurring then.

Table 3 Time (in hours) for each task by phase

	Coding	Designing	Documenting	Modeling	Reading	Rework	Testing	Other	Grand Total
Phase 1	65.58	5.55	38.40	0.00	15.50	1.27	0.00	0.83	127.23
Phase 2	18.80	11.30	29.25	0.00	4.00	0.47	0.00	0.67	64.48
Phase 3	30.55	4.22	32.63	6.52	2.33	3.65	4.35	0.00	84.25
Grand Total	114.93	21.07	100.28	6.52	21.83	6.39	4.35	1.50	275.97

2.3 Lessons Learned

2.3.1 Writing a Valid Scrolling Client

The Java Swing JScrollPane requires that any client update its preferred size and call revalidate() prior to calling repaint(). This change fixed a bug with scrolling a zoomed image.

2.3.2 The Rational Unified Process Works

I learned that executing the Rational Unified Process indeed increases the confidence one should place in plans as the project unfolds. The highly uncertain first phase gave way to confident execution of phases two and three.

3 Products

This section reviews the products of the project for quality and for possible improvements in the future.

3.1 Quality

3.1.1 Rework Ratio

The rework ratio defined is as:

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

where $E_{Defeects}$ is the effort spent fixing defects and $E_{Development}$ is the effort spent developing code. Figure 6 below shows the plot of Rework Ratio over time throughout this project. As expected, since the majority of testing occurred at the end of the project the ratio increases as the majority of defects are found. I believe that continued use of the GMoDS Test Driver and Visualizer would lead to a Rework Ratio that declines to near zero over time.

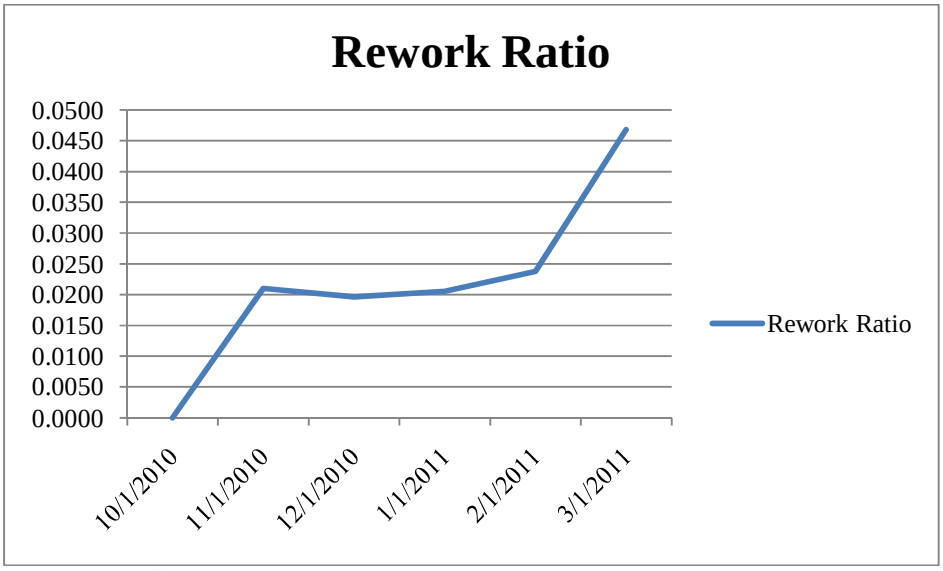

Figure 6 Rework Ratio

3.1.2 Mean Time between Failures

[1] defines Mean Time between Failures (MTBF) as the average usage time between software faults and states that "In rough terms, MTBF is computed by dividing the test hours by the numbers of type 0 and type 1 SCOs". In my case, I used the cumulative number of test hours divided by the cumulative number of faults.

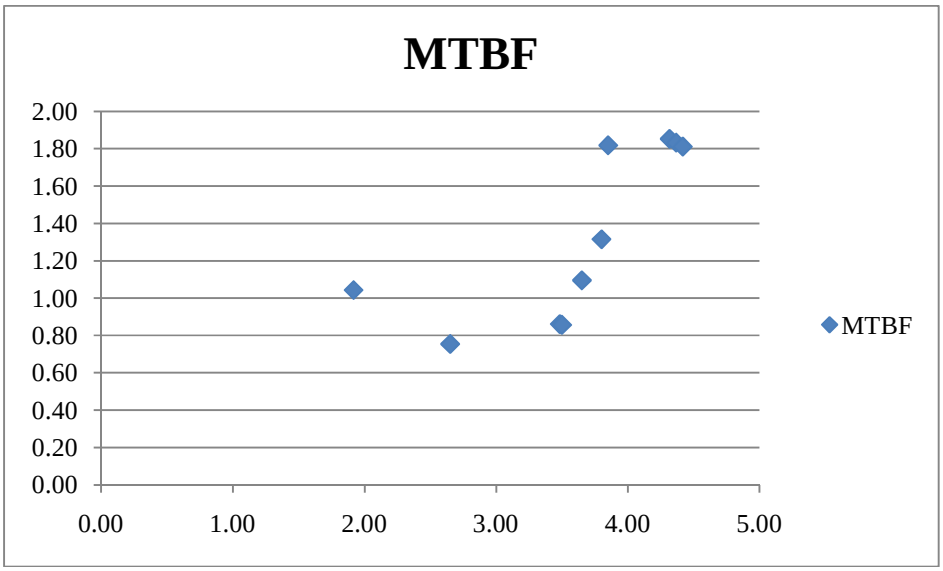

Figure 7 Mean Time between Failures

Figure 7 above shows a rising MTBF as testing begins and a bit of leveling off at the end as no more defects are found. I believe that MTBF will increase as the GMoDS Test Driver and Visualizer are put into production use.

3.2 Future Work

This section lists some future work that could improve the GMoDS Visualizer and Test Driver.

3.2.1 Dynamic Specification Tree Display

The GMoDS ChangeManager interface can notify the GMoDS Visualizer of changes to the specification tree similar to changes to the instance tree. It would not be difficult to change the Visualizer to respond to these changes by redrawing the specification tree.

3.2.2 Display Events Executed in the GMoDS Test Driver

The GMoDS Visualizer UI could be enhanced to display the list of events as they issued by the GMoDS Test Driver to GMoDS. This could improve the feedback and learning about GMoDS a user of the Test Driver experiences.