

CITS4201: Project Proposal

October 23

2008

Project Supervisor: A/Prof Terry Woodings
Author: Jackson W. Lee
Email: leej08@student.uwa.edu.au

**Studies on
Software
Comprehensibility
Standards**



Table of Contents

1.0	Introduction.....	3
1.1	Background Material.....	4
1.2	Motivations: The specific Problem.....	4
2.0	The issues this project will address.....	5
2.1	List of open questions that my work will address.....	5
2.2	The Aims I wish to achieve	5
3.0	A plan for completing the work in the given time-frame	5
	References	8

1.0 Introduction

Software maintenance can be considered as the foremost activity in a software lifecycle. It could take up to 90% of typical software products and consume as much as 70% of the lifecycle costs of a software system (Robillard 2007). Moreover, as a piece of software evolves, it can result in increasing the difficulty of maintaining it due to the changes in source code and individual programming style.

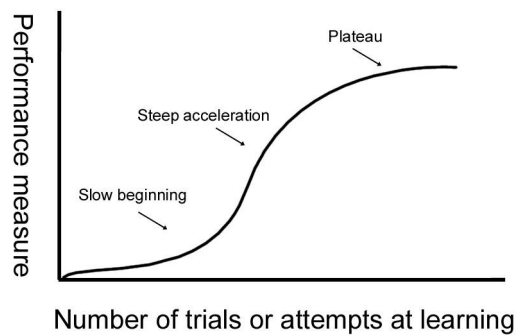


Figure 1: Traditional Learning Curve (Dewey, 2008)

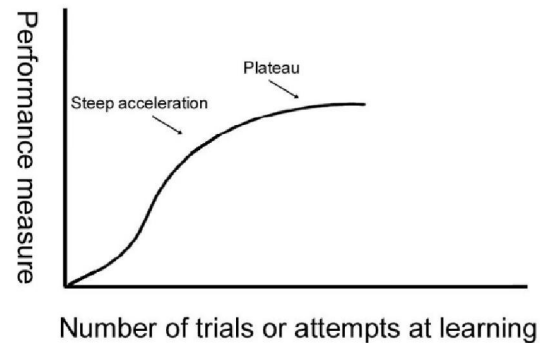


Figure 2: Possible Learning Curve with Improved Comprehensibility

Software Maintainers go through a typical learning curve for each new program or part of a program that they maintain. It is natural to have a slow beginning for most people endeavoring to learn a new system or skill. However, for software maintenance, the amount of time for a maintainer to be in the “slow beginning” portion of the learning curve can be affected by how comprehensibly the source code has been documented. Thus bringing up the problem of measure for comprehensibility, an objective software comprehensibility measure is difficult to attain due to the different understandability variables contributing to the total understandability of a piece of software:

- Documentation.
- Software structure
- Components
- Data
- Source Code

From recent research (Jin-Cherng Lin 2006), the authors have proposed a fuzzy model to integrate the separated measures of understandability. This model is further evaluated (Jin-Cherng Lin 2008) with tests on measuring understandability with the proposed fuzzy model. This project will be attempting to apply this concept and produce a standard.



1.1 Background Material

The material that has been used for background study is Software Metrics by Fenton, and other IEEE journals as referenced at the end of this document. Primarily, the main topic of study is the fuzzy model of understandability and additional background for Spatial Complexity, Number of Concepts and Typographical Style (Mohan 2004; Gupta 2006). Plus a number of other metrics like Halstead's Complexity, Cyclomatic Complexity, Cognitive Weights(Wang 2003) and Cloze Code Comprehension Tests.(Entin 1984)

1.2 Motivations: The specific Problem

The problem that I'm hoping to address is the missing standard for software code comprehensibility. The bulk of software costs in the software lifecycle is substantially large and the core activities of software maintenance is Adaptive, Corrective and perfective maintenance by modifying existing code(Mohan 2003). Thus maintainers have the need to analyze in depth any system that needs to be modified (Robillard 2007) and hence they will have to reiterate the learning process for different systems or subsystems of existing code. A maintainer will have to go through the typical learning curve and experience the "slow beginning" in every maintenance exercise. The time spent in the "slow beginning" phase of learning can vary, however it is an unchecked variable that is contributing to the overall cost of software maintenance.

If we have a standard measure for code comprehensibility, we will then be able to set quality benchmarks for programmers to adhere to. This will gradually improve the quality of software in terms of comprehensibility, maintainability, readability and perceived value(Erdogmus 2007). In turn, the time for a software maintainer to learn the new piece of code can be shortened and may look like figure 2.



2.0 The issues this project will address

2.1 List of open questions that my work will address

Can the Cloze comprehensibility Test be improved upon to give a proper measure of comprehensibility?

Can the model proposed by (Jin-Cherng Lin 2006) be used to set a standard? If so, what level should it be set at?

Can the code comprehensibility standard be applied to Java?

2.2 The Aims I wish to achieve

- A standard measure for software comprehensibility that can be applied to Java.
- Demonstrate that software with a high comprehensibility measure can reduce the time required to understand a new software system for maintenance purposes.
- Establishing a quality benchmark that can be applied to Java.

3.0 A plan for completing the work in the given time-frame

The current plan for completing the work in the given time-frame of 2 semesters is as followed:

In the first semester, background study on software quality and research on past software understandability metrics, current software understandability metrics and the state of the art of software understandability metrics. Conduct interviews with university professors that teach programming regarding their measures of code comprehensibility.

Learn and conduct measures on code samples.

In the second semester I aim to finalize a standard for measuring comprehensibility and attempt to set a quality benchmark and to do so, I will be doing the following activities.

Measure the above using known code samples from available teaching resources. Getting their related comprehensibility rating, this result is expected to be high or edited to produce high results. (Artifact A)

Conduct experimental measures using low/bad comprehensibility rating (badly commented, no naming convention). Getting their related comprehensibility rating, this result is expected to be low or (purposefully edited to produce lower results). (Artifact B)

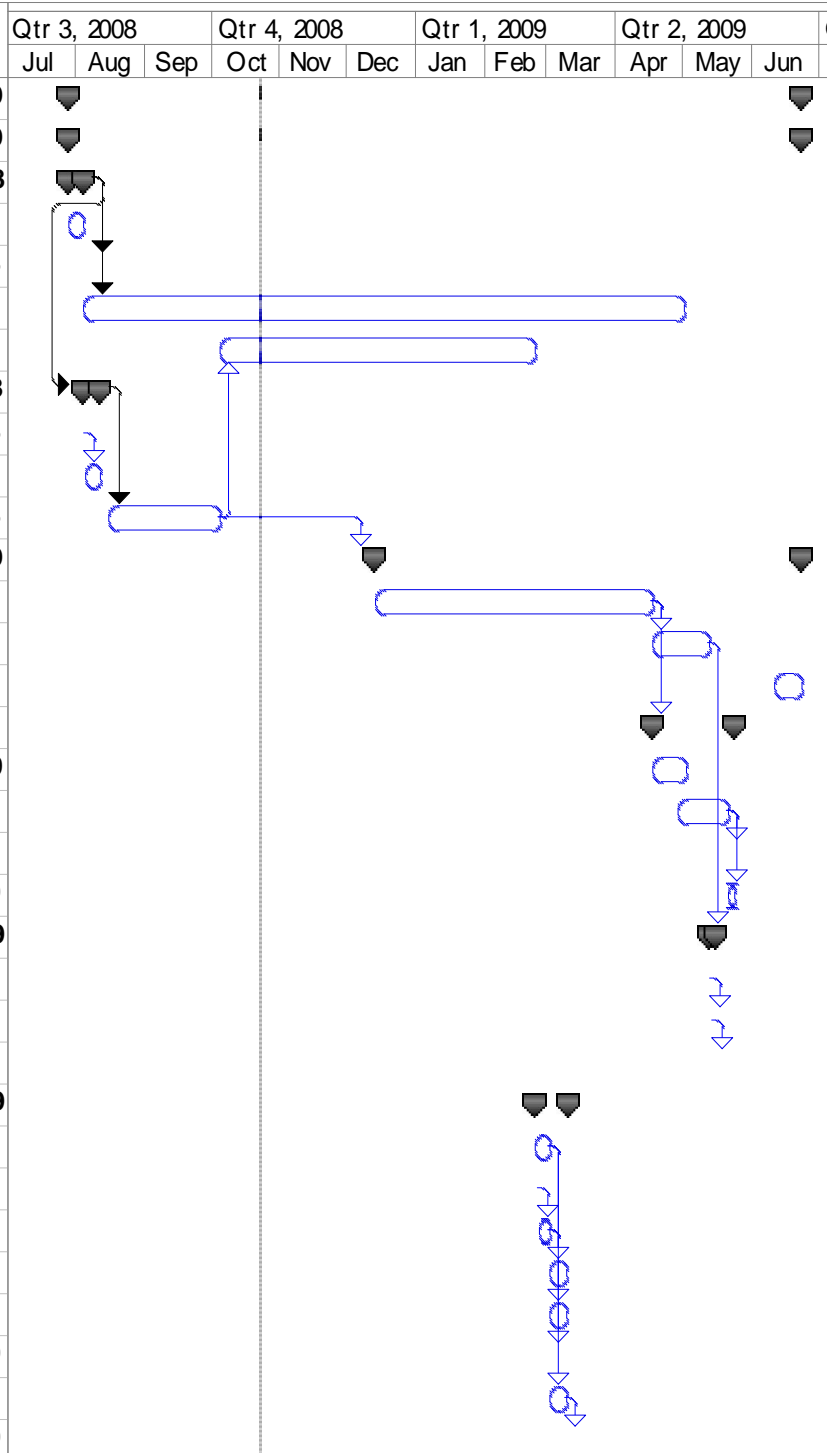
Conduct time measures on a programmers' time taken to understand artifact A and artifact B, by timing how long the programmer takes to fully understand the code and giving him a cloze



test after he is satisfied with understanding the initial code. Then measuring the percentage the programmer can reproduce from his understanding. (Entin 1986)

Following this page is a work breakdown structure on the tasks that are to be completed to finish the project on time and the tasks that has been completed or started prior the submission of this project proposal.

ID	Task Name	Duration	Start	Finish	Qtr 3, 2008			Qtr 4, 2008			Qtr 1, 2009			Qtr 2, 2009		
					Jul	Aug	Sep	Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun
1	Software Engineering Project	330 days	Mon 7/28/08	Mon 6/22/09												
2	Research	330 days	Mon 7/28/08	Mon 6/22/09												
3	Preliminary Investigation	7 days	Mon 7/28/08	Sun 8/3/08												
4	Background Reading	7 days	Mon 7/28/08	Sun 8/3/08												
5	Interviews with Lecturers	2 days	Mon 8/4/08	Tue 8/5/08												
6	Literature Review	270 days	Mon 8/4/08	Thu 4/30/09												
7	Learning and Conducting Software Metrics Measures on S	142 days	Sat 10/4/08	Sun 2/22/09												
8	Initial Project Proposal and Summary	7 days	Mon 8/4/08	Sun 8/10/08												
9	Draft Project Proposal and Summary	1 day	Mon 8/4/08	Mon 8/4/08												
10	Revising Draft Project Proposal and Summary	6 days	Tue 8/5/08	Sun 8/10/08												
11	Revised Project Proposal and Summary	50 days	Fri 8/15/08	Fri 10/3/08												
12	Dissertation	192 days	Sat 12/13/08	Mon 6/22/09												
13	Draft Dissertation	125 days	Sat 12/13/08	Thu 4/16/09												
14	Revising Dissertation into Final Dissertation	25 days	Fri 4/17/09	Mon 5/11/09												
15	Correcting the Marked Final Dissertation	12 days	Thu 6/11/09	Mon 6/22/09												
16	Seminar	37 days	Fri 4/17/09	Sat 5/23/09												
17	Prepare Seminar Title and Abstract	15 days	Fri 4/17/09	Fri 5/1/09												
18	Prepare Seminar Content	23 days	Tue 4/28/09	Wed 5/20/09												
19	Prepare Seminar Handouts	1 day	Thu 5/21/09	Thu 5/21/09												
20	Rehearse Seminar Presentation	3 days	Thu 5/21/09	Sat 5/23/09												
21	Poster	3 days	Tue 5/12/09	Thu 5/14/09												
22	Design of Draft Poster	1 day	Tue 5/12/09	Tue 5/12/09												
23	Getting Supervisor's Feedback	1 day	Wed 5/13/09	Wed 5/13/09												
24	Revise Draft Poster	1 day	Thu 5/14/09	Thu 5/14/09												
25	Measurement Exercises	15 days	Mon 2/23/09	Mon 3/9/09												
26	Designing the Measurement Exercise	7 days	Mon 2/23/09	Sun 3/1/09												
27	Sourcing for Artifacts to Measure	2 days	Mon 2/23/09	Tue 2/24/09												
28	Conducting Analysis on Artifacts	5 days	Wed 2/25/09	Sun 3/1/09												
29	Editing the Artifacts for Measurement Exercise	7 days	Mon 3/2/09	Sun 3/8/09												
30	Preparing Measurement Exercise Handouts	7 days	Mon 3/2/09	Sun 3/8/09												
31	Preparing Post Exercise Survey Forms	1 day	Mon 3/2/09	Mon 3/2/09												
32	Conducting Measurement Exercise and Tabulating Results	7 days	Mon 3/2/09	Sun 3/8/09												
33	Results Analysis	1 day	Mon 3/9/09	Mon 3/9/09												



References

- Entin, E. B. (1984). Using the Cloze Procedure to assess program reading comprehension. Fifteenth SIGCSE technical symposium on Computer science education, ACM New York.
- Entin, E. B. (1986). Using the Cloze Procedure with Computer Programs: A deeper look. Proceedings of the seventeenth SIGCSE technical symposium on Computer science education, Cincinnati, Ohio, United States.
- Erdogmus, H. (2007). "What's good software anyway?" IEEE Software **24**(2): 3.
- Gupta, J. K. C. V. (2006). "Towards Spatial Complexity Measures for Comprehension of Java Programs." Advanced Computing and Communications, 2006. ADCOM 2006. International Conference: 430 - 433
- Jin-Cherng Lin, K.-C. W. (2006). A Model for Measuring Software Understandability. Computer and Information Technology, 2006. CIT '06., IEEE CNF.
- Jin-Cherng Lin, K.-C. W. (2008). Evaluation of Software Understandability Based on Fuzzy Matrix. Fuzzy Systems, 2008. FUZZ-IEEE 2008. (IEEE World Congress on Computational Intelligence), . IEEE International Conference, IEEE CNF.
- Mohan, A. G., N.; Layzell, P.; (2004). An Initial Approach to Assessing Program Comprehensibility using Spatial Complexity, Number of Concepts and Typographical Style. Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04), IEEE.
- Mohan, N. G. A. (2003). "A Framework for Understanding Conceptual Changes in Evolving Source Code." IEEE Computer Society: 9.
- Robillard, P.-N. K., N.; Tapp, M.; Hmima, H.; (2007). Outsourcing Software Maintenance: Processes, Standards & Critical Practices. Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference, Canada, IEEE.
- Wang, J. S. Y. (2003). A new measure of software complexity based on cognitive weights. Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference Canada, IEEE CNF.
- Russ Dewey, "The Learning Curve | in Chapter 07: Cognition | from Psychology", http://www.psywww.com/intropsych/ch07_cognition/learning_curve.html , Last Accessed August 12th 2008