Affects on Maintenance of Web Software Applications

Jane Huffman Hayes
Laboratory for Advanced Networking
Computer Science
University of Kentucky
301 Rose Street
Lexington, KY 40506-0495
(859) 257-3171 (859) 323-3740fax
hayes@cs.uky.edu
and Science Applications International Corporation
jane.e.hayes@saic.com

Abstract

With our attention focused on an important question, "How can we build higher quality software that is maintainable?," we find ourselves distracted by many new developments in our world. Repeatedly we have had to take a step back, analyze a new technology, and see if all our software engineering knowledge, methods, and tools still applied or needed some refitting and retooling. The Web is definitely one of the new developments that warrants examination, must we adjust our software engineering tools and methods to make them effective for web software development? In particular, the impact of web-based development on the quality and maintainability of software is of interest.

We performed a pilot experimental study to look at the impact of web-based development on the quality and maintainability of software. The University of Kentucky (UK) is developing a new software engineering curriculum. The first course, a graduate-level survey of Software Engineering, was offered this past semester. As a major part of the course, eleven teams of students undertook a real-world project to develop a phenylalanine (phe) milligram tracker. The product, developed to run on a personal digital assistant (PDA), allows phenylketonuria (PKU) disease sufferers to monitor their diet as well as assists PKU researchers to collect data. We hypothesized that the non-web-based solutions would be more maintainable (have less defects, have lower coupling and higher cohesion, have lower complexity and be more understandable to a maintainer, etc.). However, our preliminary analysis shows that the web-based application was more maintainable than the other applications.

Keywords

software engineering; web-based software development; software maintainability; software maintenance; software quality; metrics

1 INTRODUCTION

The Web is now an accepted part of our daily work and even personal lives. This technological development affects how we work, how we shop, how we communicate with one another. Now that we have come to rely on the multi-faceted and multi-functional Web, it is important that its software perform reliably. Also, it is important that its software can be easily and reliably modified. This paper discusses a pilot experimental study that was undertaken at the University of Kentucky to examine the effects of maintenance on web software applications.

1.1 Real-World Project – Phenylalanine Tracker

The University of Kentucky (UK) is developing a new software engineering curriculum and offered the first course (CS 650) this past semester. The course emphasized maintainability and reliability of developed software and artifacts through a real-world group project. For example, students were required to collect data for maintainability and reliability growth models, to perform extensive coverage testing, and to make a major modification to their product at the end of the semester. In addition, the student teams implemented their products using web-based or non-web-based solutions. The students developed a phenylalanine (phe) milligram tracker to allow phenylketonuria (PKU) disease sufferers to monitor their diet as well as assist PKU researchers to collect data.

Working together with the UK Clinic (part of the UK Medical School), the student's projects will be used in a pilot study in the near future to determine if such an application can advance PKU research and/or assist PKU sufferers. About one in every 15,000 infants born in the United States has the inherited (genetic) disorder PKU. People who are born with PKU are normal in every way except that to stay healthy they must follow a strict diet. The diet limits phe, a common part of most foods. If phe levels in the blood of a PKU sufferer stay too high for a long time, the damage to the brain is severe and irreversible [5].

1.2 Paper Organization

The paper is organized as follows. In Section 2, related work in software maintainability and web-based software development is discussed. Section 3 discusses the semester-long Phe Tracker study, including a description of the pilot study design, our hypotheses, and the results. Finally, Section 4 presents conclusions and directions for future work.

2 BACKGROUND AND RELATED RESEARCH

Software maintainability is defined as "the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment" [6]. Maintained software refers to software systems or components that are currently in use and are being modified (we include perfective, adaptive, and corrective maintenance) on a regular basis. Many researchers are addressing software maintenance issues including program comprehension; code decay; change impact analysis, location, and propagation; maintenance processes; maintainability measurement; etc. Program comprehension is key to allowing modifications of code by other than the original programmer. Thread analysis [3] has been proposed for understanding object-oriented software, with experimental results showing that such analysis effectively supports program behavior understanding. Also, the proposed testing technique can be exploited to better test an object-oriented system. Dependency analysis, or understanding how components interact with each other to ensure all necessary changes have been made, has also been well researched. For example, Zhifeng and Rajlich present an approach that examines the impact of hidden dependencies on the process of change propagation and an algorithm that warns about the possible presence of hidden dependencies [7].

Code decay, or deterioration of the structure of code over time, is a phenomenon that has been hotly debated. Eick et al examined millions of lines of code and found mixed, but persuasive, statistical evidence of code decay, which is corroborated by developers of the code [4]. The author hypothesizes that it is increased complexity with each modification that leads to code decay. Along these lines, Banker, Datar, Kemerer, and Zweig found that complexity does make software harder and more costly to maintain [1], but did not examine the notion of increased complexity with each modification.

No references on maintenance of web-based software were found, but it is clear that we face additional challenges in maintaining such software. Web-based software has decreased software development cycle times, challenging maintainers to modify running software (operating 24x7), increased application complexity and performance requirements, etc. Similarly, the increased demand for web software and a shortage of adequately trained personnel has further contributed to poor software maintainability.

3 PHENYLALANINE (PHE) TRACKER STUDY

This section examines the experimental study undertaken in the UK software engineering course. The experimental study design, the hypotheses, and results will be presented.

3.1 Classification of Phe Tracker Project

Our motivation was to engineer a product for easy modification as well as to better understand the maintenance process. We used the real-world course project/study framework [5] to design and classify the study. The framework consists of numerous elements that are required to ensure that a project course can be used effectively as an experimental study. The classification of the Phe Tracker project per [5] is shown below.

We undertook a project whose purpose was to predict the size and effort to build the application (the product, i.e. object) as well as to implement the problem solution. We did so from the perspective of the developer, maintainer, and user advocate. The product was examined in a replicated project study (scope), where 33 software engineers, student through professional from the software engineer domain, developed one software system from the program/project domain. It was quality of life importance with real-world end users. The software system developed belonged to the medical problem domain in the nutrition monitoring system problem class (project design). It was developed with no specific experimental design.

Objective measurement of the engineering processes was performed in several criteria areas: size estimation effectiveness, complexity, fault detection, reliability growth, the relationship of design characteristics to maintainability, the relationship of maintainability to reliability, the relationship of solution type (web versus non-web) to maintainability and reliability. Preparation included artifact development (narrative statement of scope) and object development (development lifecycle steps to be followed). Execution was broken into three phases and incorporated manual monitoring of activity. Evaluation included the application of qualitative criteria and peer-project comparison. Analysis, interpretation context, extrapolation, and impact are still in progress.

3.2 Preliminary Analysis of the Phe Tracker Experimental Study

The student teams developed the Phe Tracker application according to specifications provided by the instructor (elicited from the UK Medical School and actual PKU sufferers). Phases I and II of the project embodied the initial development of a Phe Tracker application. Phase III required a major modification to the functionality of the application (for example, the students added the ability to store, retrieve, and extrapolate phenylalanine information for food by serving size). The resulting applications and software engineering artifacts were quite impressive. For example, one team, Team 8, developed a product as opposed to just a program, complete with installation disks, context-sensitive help, a professional interface and logos, etc. Their product logo (displayed during the Setup process) is shown in Figure 1.

The project is classified as an experimental study in section 3.1. There were three main hypotheses being evaluated by this small study [5]: (1) maintenance on a product decreases its quality, (2) students improve their estimating skills over time, and (3) solution paradigm (web, non-web, programming language, etc.) has no impact on quality.

As the real-world project/study approach was just evolving during the CS 650 course, the experimental study was not handled with rigor or formality. For example, students selected their own teams (no controlling for effect of some teams having more experienced or talented members than others), there were only 11 teams (small sample), and teams were permitted to select the programming language (nine selected Java, one selected C++, one selected VisualBasic) and paradigm (ten selected non-web-based, one selected web-based). Also, many teams did not supply all requested data for all phases of the project. The students all worked to the same specifications and deadlines. Recall that implementation occurred in Phase II with modifications/maintenance in Phase III.

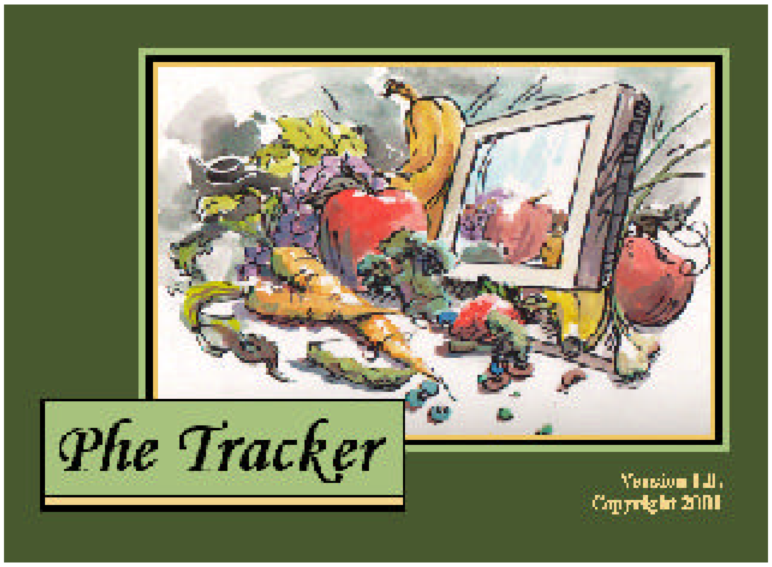

Figure 1. Team 8 product logo.

Examining the first hypothesis, we looked at several sub-hypotheses: weighted methods per class (WMPC) [2] will increase after maintenance occurs, complexity will increase after maintenance occurs, number of defects will increase after maintenance occurs, coupling will increase after maintenance occurs, cohesion [2] will decrease after maintenance occurs, and size will increase after maintenance occurs. Our small study found no such trend for WMPC (three teams had higher values after maintenance, four did not), complexity did increase (but only three teams reported data before and after maintenance), number of defects decreased for maintenance (all four teams reporting Phase II and III data had less defects in Phase III than in Phase II), and coupling was inconclusive (increased for two teams, decreased for two teams) as was cohesion. There were more function points (FPs) and lines of code (LOC) after maintenance (four of six teams had an increase in FPs, six of seven teams had an increase in LOCs). With such mixed results, we cannot say anything conclusive from our study about the relationship between maintenance and quality.

Looking at the second hypothesis, we had two sub-hypotheses: FP estimates will improve over time, and LOC estimates will improve over time. Our study found that 10 of the 11 teams improved their FP or LOC estimating. This finding suggests that estimation skills will improve with experience. A related hypothesis was that students using FPs would make better estimates than those using LOC. Indeed, this was the case. All six of the teams reporting FP estimates and actuals were very accurate in estimating, as shown in Figure 2 (the other teams used LOC and did not provide FP info). Also, all FP actuals were equal to or larger than the estimates. For LOC, two teams underestimated and four teams overestimated with several teams missing their estimates by 100%.

For the third hypothesis, we had several sub-hypotheses: non-web-based applications will have lower complexity than web-based applications, non-web-based applications will have lower coupling than web-based applications, non-web-based applications will require less effort, non-web-based applications will have less defects, Java applications will have lower complexity than non-Java applications, Java applications will have lower coupling than non-Java applications, Java applications will require less effort, Java applications will have less defects. Unfortunately, we lacked the necessary data on the non-Java applications to examine these ideas. Nonetheless, we were able to examine the web-based maintainability hypotheses. These results are discussed below.

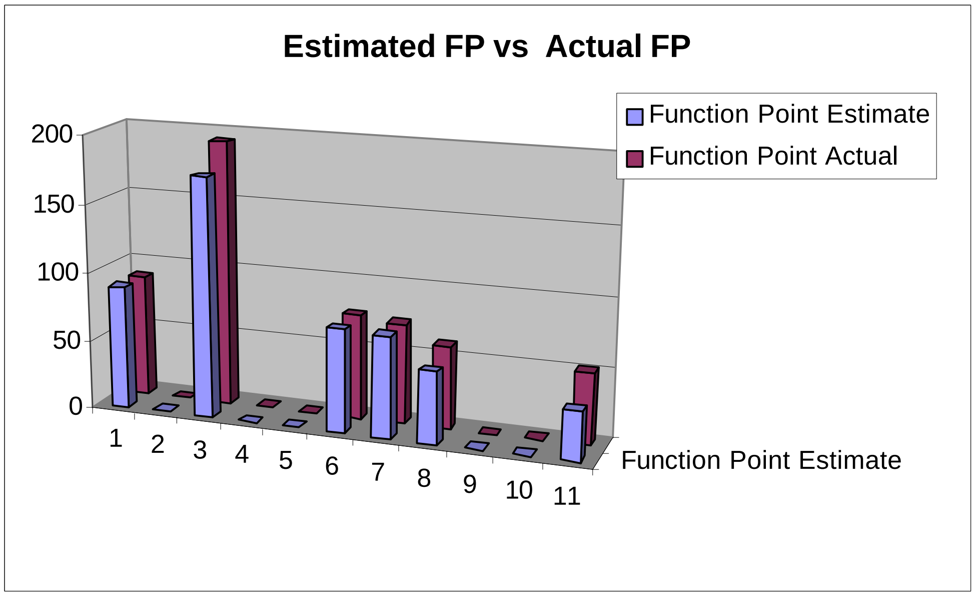

Figure 2. FP Estimation Error.

Team 11 implemented a web-based solution to the Phe Tracker problem. Their WMPC measure actually decreased from Phase II to Phase III (one way of measuring complexity). Also, Team 11's WMPC was lower than six of the seven teams reporting the measure for Phase III. It appears that the web-based solution was less complex than the non-web-based solutions. Team 11's coupling measure, coupling between objects (CBO) [2], was higher than four of the teams reporting for Phase III, but lower than three of the teams. Therefore, no conclusion can be made on coupling. Team 11's effort per function point was lower than one team and higher than two teams reporting the Phase III data. It appears that more effort may have been required per function point. Team 11's average cohesion, lack of cohesion for methods (LCOM) [2], for Phase III was significantly higher than the other two teams reporting data. This implies that the web-based solution was more cohesive. We lacked necessary data to evaluate the defect hypothesis. From these results, we think it might be the case that web-based solutions are easier to maintain than non-web-based solutions. We found decreased complexity, high cohesion, and decreased effort per function point maintained for the web-based solution, all contributing to maintainable products. Bear in mind though that we had a very small study, and it could be the case that internal and external threats to validity (e.g., perhaps the students on Team 11 were truly exceptional software engineers) were responsible for these results.

4 CONCLUSIONS AND FUTURE WORK

We undertook a small study, as part of a university course, to examine software maintainability. Specifically, we felt that web-based software would be harder to maintain than non-web-based. In contrast, our study showed evidence for the opposite, that web software is less complex, more cohesive, and requires less effort to modify than non-web software.

What do these results mean to us? To instructors or researchers in the field of software engineering, they mean that real-world course projects can be used as experimental studies also, with advanced planning and careful attention to the framework of the study/project. It means that web software may actually be more maintainable than its non-web-based counterpart. It also means that further investigation and research is required, as our study was very small and informal.

Based on our work, we offer the following suggestions to other researchers:
- select an important, real-world project and have it also serve as an experimental study
- plan to have the students make a major modification to the project to ensure design for maintainability
- try to have half the teams work on web software, half on non-web software

- ensure that all teams report all metrics for all phases
- try to collaborate with another department or school at your institution.

There is further work to be done. Future plans include: repeating the web software maintenance study using more stringent controls, larger samples, etc.; collecting information for numerous maintainability measures appearing in the literature and evaluating their effectiveness; and possibly writing a grant to receive research funding for both the medical aspects and software engineering/maintainability aspects of this research bed and project.

ACKNOWLEDGMENTS

I would like to thank all of the students who participated in the Software Engineering course (UK's CS 650). Thanks to Team 8 (Andrew Mertz, Venkata Velagapudi, Sanjiv Ganguli, and Jessica Mertz - artist) for providing artifacts, and to Naresh Mohamed for assisting with analysis. A special thanks to Hannah and Kelly Marcum and to the UK Medical School for their participation in this project.

REFERENCES

[1] Banker, Rajiv D., Datar, Srikant M.,Kemerer, Chris F., and Zweig, Dani. Software complexity and maintenance costs. *Communications of the ACM*, 36, 11 (Nov. 1993), pp. 81 – 94.

[2] Chidamber, Shyam and Kemerer, Chris. A metrics suite for OO design. *IEEE Transactions on Software Engineering*, Volume 20, No. 6., June 1994, pp. 476-493.

[3] di Lucca, G.A., Fasolino, A.R., and de Carlini, U. An algebraic notation for representing threads in object oriented software comprehension*. Proceedings of the 9th International Workshop on Program Comprehension (IWPC 2001)*, 12-13 May 2001, pp. 176 – 185.

[4] Eick, S.G., Graves, T.L., Karr, A.F., Marron, J.S., and Mockus, A. Does code decay? Assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, Vol SE-27, No. 1, January 2001, pp. 1 – 12.

[5] Hayes, Jane Huffman. Energizing reliability engineering education through real-world projects. Submitted to *Conference on Software Engineering Education and Training (CSEE&T) 2002*.

[6] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

[7] Zhifeng Yu, and Rajlich, U. Hidden dependencies in program comprehension and change propagation. *Proceedings of the 9th International Workshop on Program Comprehension (IWPC 2001)*, 12-13 May 2001, pp. 293 – 299.