THE EFFECT OF SEMANTIC KNOWLEDGE ON SOFTWARE MAINTENANCE: AN EMPIRICAL STUDY

Sam Ramanujan, University of Central Missouri, ramanujan@ucmo.edu
Someswar Kesh, University of Central Missouri, kesh@ucmo.edu

ABSTRACT

A model for software maintenance based on the Human Information Processing (HIP) model has been developed and tested. Based on the results, recommendations for further research as well as for practitioners have been made.

Keywords: software maintenance, HIP, semantic knowledge, programmer characteristics

INTRODUCTION

Software maintenance is an important, resource consuming and expensive task. Therefore proper resource allocation to maintenance activities is extremely important. In this research, a model for software maintenance that can assist in resource allocation for maintenance activities has been presented. Particular emphasis has been placed on the semantic knowledge of programmers. Based on this model, hypotheses relating to the programmer's semantic knowledge have been developed and tested.

THEORETICAL FOUNDATIONS OF THE RESEARCH

Components of the HIP Model

This study uses the Human Information Processing (HIP) model as the theoretical basis. The main components of the HIP model are: sensory registers, short-term memory (STM), long-term memory (LTM) and buffer. Sensory registers exist for each of the senses and have been given a variety of names by psychologists; sense-information stores, iconic stores, and precategorical stores. The icon and echo are the two most extensively used sensory registers and they correspond to our visual and auditory senses respectively. This study focuses on the visual sensory register, icon. In buffer memory, the information from STM and LTM are integrated to build new structures to generate a semantic structure for a problem's solution. This semantic structure is called internal semantics. In case of learning, this semantic structure is stored in LTM for further use (Ramanujan and Cooper, 1994).

HIP Perspective on Software Maintenance

The program maintenance task can be broken into three sequential subtasks, (i) program comprehension (ii) program modification and (iii) program composition. From a HIP perspective, these are explained as follows:

Program Comprehension: During this stage, the maintenance programmer creates a multilevel semantic structure in the buffer of the problem using existing syntactic and semantic structures in the programmer's LTM.

Program Modification: Program modification is done in the buffer by synthesizing the semantic structures of the existing programs and the required modification(s). The output at this stage is a semantic structure that constitutes the program specifications of the modified program.

Program Composition: When a program composition task is presented to the maintenance programmer, the program specifications arrive in the programmer's buffer through STM. The program is then analyzed and represented in terms of a "given state" and a "desired state" (Wickelgren, 1974).

Characteristics of the Maintenance Task

The dependent variable used here is maintenance effort. In this study, the dependent variable is the time required to successfully maintain a program since one of the components of quality, semantic knowledge of programming structures, serves as an independent variable. From a HIP perspective, maintenance effort is determined by programmer characteristics, program characteristics (Ramanujan and Cooper, 1994, Alain 2002), and organizational characteristics (Rainer 2003, Boehm-Davis 1992,). These form the foundation for identifying the independent variables in this study

Programmer Characteristics

The primary programmer characteristic in maintenance is the semantic knowledge of the programmer (Joergensen 2004, Rainer 2003, Joergensen and Soerberg 2002). Programmers possess high or low level semantic knowledge which

characterizes them as an expert or a novice. In this research an instrument that classifies programmers into experts and novices based on such factors such as the size of the program, the variations in control flow structures and the number of programs maintained has been used. The characteristics of semantic knowledge based on programmer's experience are also captured because theory suggests that the type of experience dictates the level of semantic knowledge.

Program Characteristics

The program characteristics considered are, indentation, variable name mnemonicity, comments, modularity, program size, and complexity of control flow. Studies by Vessey and Weber (1984) show that indentation reduces program effort. Similar results were found for the other variables as well (Robbilard et. al, 2004, Ko et. al, 2006).

Organizational Characteristics

Organizational Characteristics used are the effect of time pressure, and request characteristics in terms of the size of modification. Based on these, the following hypotheses were tested.

HYPOTHESES

Hypothesis 1:

Subjects with a high level of semantic knowledge will maintain a program in significantly less time when compared to subjects with a low level of semantic knowledge.

Hypothesis 2:

The difference in time taken to maintain a program with a high level of control flow complexity and a program with a low level of control flow complexity will be significantly greater for subjects with a low level of semantic knowledge.

Hypothesis 3:

For subjects with a low level of semantic knowledge, the time taken to maintain small programs with a higher level of complexity will not be significantly greater than the time taken to maintain small programs with a lower level of complexity.

Hypothesis 4:

The difference in time taken to maintain a program under a low level of time pressure and the same program under a high level of time pressure will be significantly greater for subjects with a low level of semantic knowledge when compared to subjects with a high level of semantic knowledge.

Hypothesis 5:

The difference in time taken to maintain a program with a low level of repair request detail and the same program with a high level of repair request detail will be significantly greater for subjects with a low level of semantic knowledge when compared to subjects with a high level of semantic knowledge.

RESEARCH PLAN AND EXPERIMENTATION

Experimental Environment and Methodology

This research is an empirical evaluation for the hypotheses mentioned. Students from "C" and object-oriented programming courses offered in the College of Business Administration and Department of Computer Science at the University of Houston served as subjects. The experiment was designed to study the effect of independent variables described in the previous section and some of their interactions in the maintenance effort. The experimental tasks involve the maintenance of programs in the 'C' programming language.

Data in the experiment was collected using the Program Maintenance Performance Testing System (PROMPTS). For each program shown to the subject, PROMPTS records the total of the time required to read the program, complete the task successfully and maintain the program. Once PROMPTS is invoked the "Introduction Screen" follows the personal details screen and explains how to perform the maintenance tasks using PROMPTS. The subject can then choose one of the three actions: (a) proceed to the task of maintaining the program (b) suspend the experiment for a few minutes or (c) suspend the experiment for an indefinite period of time.

If the subject chooses to continue with the experiment, PROMPTS displays a "program window" containing a program that requires maintenance. Programs that are designated as challenge programs had a red border around it. When the program has been correctly modified by replacing all erroneous program lines with correct program lines from the choice window, PROMPTS informs

the subject and invokes the introduction screen for the next program. When the program is the last program in the experiment, the termination screen is invoked with a message thanking the subject for participation in the experiment and providing the contact address for the researcher. For each program shown to the subject, the system records the time required to read the program, read the task and successfully maintain the program (i.e., choose the correct replacement lines from the choice window). If the subject is unable to maintain the program in a reasonable amount of time (as determined by the pilot study) the researcher can intervene and allow the subject to proceed to the next maintenance task. Subjects are also provided with two maintenance tasks in order to train them with the operation of the PROMPTs software.

Design and Experimental Procedure

The research design used in this experiment is a variation of the multi-group posttest design with multiple treatments. The treatments include (a) variations in program size (b) variable name mnemonicity (c) control flow complexity (d) level of documentation (e) time pressure and (f) repair request detail. The time taken to correctly maintain a program is the dependent variable and serves as the posttest measure. This kind of design provides a certain degree of protection to the research when a true experiment (random assignment of subjects to treatment groups) inadvertently degenerates into a quasi-experimental design when the randomization is violated or contaminated by conditions in the research environment not under the control of the experimenter. The multi-group posttest design was used to control for threats to internal validity and various measures like not imposing a penalty for poor performance was used to control for threats against external validity.

DATA ANALYSIS AND RESULTS

Characteristics of the sample

The sample consists of 100 subjects. One group of fifty subjects came from an introductory "C" language course. The second group of fifty subjects came mainly from various companies such as Software Interfaces, RCG Information Technology etc. and from a senior class in the Computer Science program at the University of Houston.

The subjects were categorized into two groups, those with low semantic knowledge and those with high semantic knowledge. An instrument was used to classify subjects. This was based on the programmer's programming experience, program maintenance experience, knowledge of programming structures, formal training in programming and systems development and level of programming experience. A composite score was computed for each subject based on the factor weights and the subject's response to the 16 questions in the instrument. A t-test conformed that these two groups are significantly different with a p-value of .0001.

The 19-item questionnaire used in this study to classify subjects into high or low semantic knowledge groups had a reliability of 0.98 as measured by Cronbach's alpha. The hypotheses suggested in the previous section were tested using the Analysis of Variance (ANOVA) procedure. In order for ANOVA to apply to a set of data, two conditions must be fulfilled (1) scores must be normally distributed in the population (2) the variance in the treatment conditions or groups must be homogeneous. Though in most cases violation of these assumptions does not severely affect the outcome of the ANOVA procedure, the assumptions were nonetheless tested in an effort to reflect the rigor of the analysis. Shapiro-Wilk's test showed the data to be normally distributed. ANOVA is robust to deviations from the equality of variance assumption provided all groups have the same number of assumptions. In this research, the studentized residuals were used to test for homogeneity in variances between groups. Since the studentized residuals were between –2 and +2 for almost all observations, it can be inferred that the groups have equal variance.

Results of the study

In this section, the results of ANOVAs are presented for the hypotheses presented previously. Hypothesis 1 suggests there is a significant difference in time taken to maintain a program depending on the semantic knowledge of the programmer involved in the maintenance. Results of the ANOVA conducted to test this hypothesis are given in Table 2 and indicate that programmers with a high level of semantic knowledge take significantly less time to maintain a program when compared to programmers with a lower level of semantic knowledge.

Table 1(a): Descriptive Statistics for Hypothesis 1

Hypothesis	n	F-Statistic	p-value
Hypothesis 1: μ(LOW SEMANTICS)-μ (HIGH SEMANTICS) > 0	800.00	27.01	.0002

Table 1(b): ANOVA/contrast results for Hypothesis 1

	n	Mean time to maintain (Seconds)
Programs maintained by low semantic group	400.00	287.27
Programs maintained by high semantic group	400.00	224.92

Support for this program suggests the use of programmers with a high level of semantic knowledge for software maintenance. Swanson and Beath (1990) found that most organizations use novices or less competent programmers for software maintenance. They suggested that the use of low quality programmers could be a reason for high maintenance costs.

Hypothesis 2 predicts that changes in control flow complexity will have less effect on time to maintain a program when a program when these programs are maintained by programmers with high semantic knowledge when compared to the same programs being maintained by programmers with low semantic knowledge.

Analysis of the data provides support for Hypothesis 2.

Table 2: ANOVA/contrast results for Hypothesis 2

Hypothesis	n	T-statistic	p-value
Hypothesis 2: (μ(prog11+prog12+prog31+prog32)-μ(prog21+prog22+prog41+prog42)) low semantic > (μ(prog11+prog12+prog31+prog32)-μ(prog21+prog22+prog41+prog42)) high semantic	400.00	11.90	.0001

It was found that the decrease in maintenance effort due to the decrease in level of control flow complexity was greater in programmers with low semantic knowledge when compared to programmers with high semantic knowledge. As shown in table 3, this result has a calculated statistic of 11.90($p < 0.0001$) and leads to the conclusion that the effect of control flow complexity on maintenance effort is stronger when programs are maintained by programmers with lower level of semantic knowledge. This suggests that programmers who have gained experience by working with programs with varied control flow structures should be assigned to maintain complex programs while novices could be used to maintain simpler programs. A maintenance shop can therefore use a mix of both expert and novices without reducing the overall efficiency of the group.

According to hypothesis 3, programmers with lower level of semantic knowledge will take the same time to maintain complex programs they would take to maintain a simple program. In this study, we found that there is a significant difference in the time required to maintain a small program with a high level of control flow complexity when compared to a program with a lower level of control flow complexity. The ANOVA results for the Hypothesis 3 appear in Table 3.

Table 3:ANOVA/contrast results for Hypothesis 3

Hypothesis	n	T-statistic	p-value
Hypothesis 3: (μ(prog31+prog32)-μ(prog41+prog42) low semantic knowledge = 0	100.00	224.40	.0001

Rejection of Hypothesis 3 suggests that maintenance effort is sensitive to changes in control flow complexity than expected. This hypothesis compares the time taken to maintain two 16 line programs that have control flow complexity measures of 1 and 4.5 respectively. The large magnitude in difference (4.5:1) in control flow complexity may have lead to the difference in time taken to maintain these programs. It is possible that maintaining a program with control flow complexity of 4.5 led to the formation of more than seven chunks in the STM, thereby leading to a significantly higher maintenance effort when compared to maintaining a program with control flow complexity of 1 which leads to the formation of fewer than seven chunks. This was not expected given the current practice of classifying all programs with control flow complexity (McCabe's number) of 10 and above as complex programs.

Current results suggest that even small programs with control flow complexity of 4.5 can lead to formation of more than 7 chunks in the STM and thus should be classified as a complex program.

Further research is required to study the relationship between control flow complexity and the number of chunks formed in the STM while maintaining small programs. This may help classify programs into simple and complex programs.

Hypothesis 4 asserts that for programmers with a high level of semantic knowledge the difference in time to maintain a program under low time pressure and under high time pressure will be small compared to programmers with a low level of semantic knowledge. As shown in Table 4, the results support hypothesis 4.

Table 4: ANOVA/contrast results for Hypothesis 4

Hypothesis	n	T-statistic	p-value
Hypothesis 4: (μ(prog52+prog61)-μ(prog51+prog62) low semantic knowledge > (μ(prog52+prog61)-μ(prog51+prog62) high semantic knowledge	400.00	3.26	.0012

This result suggests that the increased time pressure reduces maintenance effort to a greater extent in programmers with lower semantic knowledge. This encourages managers to consider the use of time pressure to reduce maintenance effort only when maintained by novices.

Hypothesis 5 asserts that repair request detail will have a significantly less effect on maintenance effort for programmers with a high level of semantic knowledge when compared to programmers with a low level of semantic knowledge. The results of Hypothesis 5 are presented in Table 5. It can be concluded that repair request detail has a positive effect on maintenance for programmers with lower semantic knowledge. Support for Hypothesis 5, encourages organizations to set standards for written repair requests.

Table 5: ANOVA/contrast results for Hypothesis 5

Hypothesis	n	T-statistic	p-value
Hypothesis 5: (μ(prog92)-μ(prog91)) low semantic knowledge > (μ(prog92)-μ(prog91)) high semantic knowledge	100.00	12.62	.0001

These standards should be based on the semantic knowledge of the programmers in the software maintenance group.

CONTRIBUTIONS OF THE RESEARCH

Contributions to MIS Research

The study makes several contributions to MIS research. The study describes and experimentally validates a theoretically based model to study factors affecting software maintenance effort. It provides a validated model of software maintenance and organizes prior research using this model. It illustrates the use of the model in generating propositions concerning software maintenance efforts. This, in turn, will help in building a theoretical basis for software maintenance effort. There has been little progress in software maintenance for formulating a theoretical basis for identifying and describing factors that affect software maintenance effort (Ramanujan and Cooper, 1994). This foundation can help in synthesizing empirical findings and direct attention towards empirical questions that merit investigation. For practitioners, the study supports the use of programmers who have gained experience by working with programs of varied control structures. Greater detail of repair requests also helps maintenance effort.

Limitations of the Study

First, the investigations in this research are limited to laboratory experimentation thereby limiting its external validity. Second, although a concerted effort was made to secure a representative sample, certain groups (especially college students from the 'C' programming class offered in the College of Business Administration at the University of Houston) tended to be disproportionately represented in the sample due to practical considerations. Any claim of external validity must be tempered with the fact that the sample, strictly speaking, was not randomly selected from the target population. The next limitation was, is the limited manipulation of the independent variables. All independent variables in this study are classified dichotomously as either high or low. Such dichotomous measurement allows only for relative analysis and limits the results of the study to real-world situations. Finally, the size of the tasks was dictated by the limited availability of the subjects for the experiments. Tasks were designed so that all subjects were able to complete them in a reasonable amount of time (i.e., two hours).

CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

This research shows that software maintenance requires programmers with high level of semantic knowledge, including conditions where significant complexity is involved. Moreover time pressure will work better for programmers with low level of software maintenance. Future research can take three forms. First, the application of the HIP model to software maintenance effort can be used for generating additional propositions that focus on the reduction of maintenance costs. Secondly, the SME model can be enriched by incorporating, in the HIP model, the results of future studies on long-term memory, short-term memory, buffer and sensory registers. Finally, further research can validate the propositions through field experiments. Studies of this nature will enhance the external validity of the results of this research. This study can be described as a precursor to a program of research in the area of software maintenance effort.

REFERENCES

1. Alain, A. et. al., (2002), Field Studies Using Functional Size Measurement in Building Estimation Models for Software Maintenance. Journal of Software Maintenance and Evolution: Research and Practice, 14(1), 31-64.

2. Boehm-Davis, D.A., et. al. (1992). The Role of Program Structure on Software Maintenance, International Journal of Man-Machine Studies, 36(1), 21-63.

3. Joergensen, M., (2004). A Review of Studies on Expert Estimation of Software Development Effort. Journal of Systems and Software, 70(2). 37-60.

4. Joergensen, M., and Sjoeberg, D.I.K. (2002),"Impact of Experience on Software maintenance Skills", Journal of Software maintenance and Evolution, Research and Practice, 14(2), 123-146.

5. Ko, A.J., et. al (2006), An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information During Software maintenance Tasks, IEEE Transactions on Software Engineering. 32(12), 971-987.

6. Rainer, K. (2003), Software Visualization in Software Maintenance, Reverse Engineering and Re-Engineering: A Research Survey", Journal of

Software Maintenance and Evolution: Research and Practice, 15(2), 87-109.

7. Ramanujan, S. and Cooper, R.B. (1994). An Human Information Processing Perspective on Software Maintenance. Omega. 22(2), 185-203.

8. Robillard, M.P. et. al. (2004), How Effective Developers Investigate Source Code : An Exploratory Study, IEEE Transactions on Software Engineering. 30(12). 889-903.

9. Vessey, I., and Weber. R. (1984), Conditional Statements and Program Coding: An Experimental Evaluation. International Journal of Man-Machine Studies. 21(2). 161-190.

10. Wickelgren, W. (1974) How to Solve Problems. W.H. Freeman, San Francisco.