A Mapping Study of Software Causal Factors for Improving Maintenance

Carson Carroll[1], Davide Falessi[1], Vanessa Forney[1], Alexa Frances[1], Clemente Izurieta[2], Carolyn Seaman[3]
[1]{ccarro03, dfalessi, vforney, amfranci}@calpoly.edu, California Polytechnic State University, 1+(805) 756-2344
[2]clemente.izurieta@cs.montana.edu, Montana State University, 1+(406) 994-3720
[3]cseaman@umbc.edu, University of Maryland Baltimore County, 1+(410) 455-3937

***Abstract*— Context: Software maintenance is important to keep existing software systems functional for organizations or users that depend on that software. Goal: We aim to identify the factors, i.e., software characteristics such as code complexity, leading to maintenance problems. Method: We present a Mapping Study (MS) on controlled experiments that investigated software characteristics related to defects during maintenance. Results: The search strategy identified 78 papers, of which 9 have been included in our study, dated from 1985 to 2013, after applying our inclusion and exclusion criteria. We extracted data from these papers to identify the research methods, and the independent, dependent, blocked, and measured variables. Conclusions: Our MS results point to a weak evidence on software factors causing defects during maintenance. Stronger evidence can be developed via more controlled experiments that address multiple independent variables and hold the software objects constant.**

Keywords—Software maintenance, mapping study, systematic literature review, controlled experiments, defects.

I. INTRODUCTION

A. Context

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" [1]. Software maintainability continues to be a critically important area of research. Keeping existing software systems operational and valuable for years, even decades, is necessary for the economic viability and ongoing functioning of the organizations and segments of society that depend on those systems. Examples of maintenance problems include high numbers of defects or low development speed, which in turn lead to increasing cost and decreasing quality in the maintenance phase. One important way to ensure maintainability is to avoid the creation, or remove the existing, software characteristics that are expected to lead to maintenance problems. For example, a software system can be analyzed for complexity, discovering components that can then be refactored to lower complexity, possibly facilitating future extensions to those components. A major practical hindrance to doing this successfully lies in the lack of underlying knowledge we have about the causal relationships between software characteristics and future maintenance problems.

B. Motivation

A significant amount of research exists in the software engineering literature that discusses software maintainability. Specifically we know there is a correlation between several software characteristics (e.g., god classes [2], code clones [3] [13], and in general code smells [2]) and maintenance problems; which include number of defects or decreased speed.

However, this body of knowledge focuses on correlational rather than causal relationships. Although correlational relationships support the activities of understanding, planning, and predicting software development, only causal relationships lead to maintainability improvements. Specifically, knowing that two characteristics are correlated is not the same as knowing that one of those characteristics causes the other, and by extension that changing one will have an effect on the other. This is often because there is a third characteristic that has a causal effect on both of the correlated characteristics. As a simple example, there is a well-known correlation between red hair and susceptibility to sunburn. However, this does not mean that a redhead who dies his hair black should expect improvements in sunburn protection. Similarly, removing correlational relationships does not mean that system maintainability will improve.

Consider the example of the relationship between code complexity metrics and defect-proneness. Simple code cannot effectively implement complex requirements, using Brooks' notion of essential and accidental complexity [4]. In other words, a third variable, the complexity of the problem being solved (or of the requirements) may be driving both the complexity of the code and the high number of defects. Thus, the suggestion to reduce code complexity via refactoring activities in order to reduce defects can be as misleading as marketing hair dye to redheads to prevent sunburn.

Practitioners are already using correlational, rather than causal, research results to guide their improvement efforts [5]. We believe this is perilous for several reasons. Improvement efforts in industry are expensive in most cases. Thus, it is important that they are based on empirical results that show that such improvement efforts will cause an effect; considerable resources will otherwise be wasted in efforts that will have no impact on future maintenance. Further, unsuccessful improvement efforts based on the current body of evidence will erode the industry's confidence in scholarly research, and damage the already fragile relationship between research and practice.

978-1-4673-7899-4/15/$31.00 ©2015 IEEE

In conclusion, in order to allow practitioners to employ effective improvement practices, it is imperative to build a body of evidence that reveals the causal relationships between software characteristics and maintainability problems.

C. Aim

Causality subsumes correlation, so past work focusing on correlation is necessary and relevant. We believe that this area of research has reached a level of maturity that allows a shift in focus to examination of causal rather than simply correlational relationships. Such a shift is the necessary next step towards transforming this area of investigation into actionable knowledge that can be used in software maintenance practice.

The goal of this study is to find and understand the software characteristics that play a role in producing maintainable code by developing a knowledge base about causal relationships between observable code characteristics and maintenance problems. We present a Mapping Study (MS) on controlled experiments that investigated software characteristics related to defects during maintenance. We report on the need for more controlled experiments featuring two important characteristics: 1) addressing multiple independent variables and 2) holding the experimental software objects constant. These characteristics are expensive but also vital for identifying the software characteristics causing defects during maintenance.

D. Structure

This paper is organized into five sections as follows. Section II describes the research methods used. Section III describes the journals and conferences selected. Section IV presents the MS results. Finally, Section V concludes the paper with a summary and presents future work.

II. Research Method

The research described in this paper has been carried out in March 2015 by following the Kitchenham and Charters guidelines for conducting an MS [6].

A. Research Questions

The MS was carried out using the following research questions:

1) RQ1:Do any software characteristics show a statistical impact on defects?

2) RQ2: What is the distribution of the number of independent variables that have been considered across controlled experiments?

3) RQ3: Do any software characteristics have a causal relationship with defects in maintenance?

B. Search String

We applied the query found in Figure 1, which was designed to achieve a balance between relevant results and sufficient papers for a thorough analysis. The first requirement for our paper selection was the phrase "controlled experiment" because we were searching for studies that directly measured how certain software characteristics affect maintainability. Controlled experiments are the primary way to demonstrate a causal relationship between one of these factors and maintenance outcomes. Next, we combined this with a Boolean AND operator to the keyword "software" because we only wanted papers relevant to software. Lastly, we removed any papers related to software inspection with a Boolean NOT.

(Abstract:"controlled experiment" **AND** Abstract:software **NOT** Document Title:inspection) **AND** (Abstract:bug **OR** Abstract:anomaly **OR** Abstract:anomalies **OR** Abstract:defect **OR** Abstract:failure **OR** Abstract:fault)

Figure 1: The search string used in this mapping study

After our query was narrowed down to controlled experiments within software engineering, we had an additional Boolean AND with search terms related to software maintenance. We focused on maintainability in reference to defects in order to limit the scope of the results. These terms include the following: bug, anomaly, anomalies, defect, failure, and fault.

We also applied snowballing. Specifically, in addition to those returned using the search string, one other paper was selected because it was a replication of a selected paper [72]. This paper was not returned in the search because "controlled experiment" was in the title and not the abstract, which is a potential limitation of our query. Another paper was added after reading through the sources from one of the selected papers because it was relevant to our research. This search strategy resulted in a total of 78 'hits' that included 9 precisely relevant papers, after applying inclusion and exclusion criteria.

III. Sources

We selected the journals and conferences most relevant to our area of research. They included:

- Journal of Systems and Software,
- Empirical Software Engineering and Measurement
- Information and Software Technology,
- ACM Surveys,
- Software Quality Journal,
- IEEE Transactions on Software Engineering
- International Conference on Software Engineering
- Transactions on Software Engineering and Methodology
- Foundations of Software Engineering
- IEEE Software,
- Replication in Empirical Software Engineering Research
- Symposium on Software Engineering for Adaptive and Self-Managing Systems
- International Conference on Program Comprehension
- Working Conference on Reverse Engineering (WCRE), and
- Software IET.

We remind that papers belonging to additional sources have been included via snowballing.

A. Inclusion and Exclusion Criteria

Papers resulting from the database query were filtered based on a set of inclusion and exclusion criteria, described in Table I. These criteria were selected based upon past research related to our topic and advice from software maintenance experts.

TABLE I. Criteria used to accept or reject each paper

Inclusion Criteria	• Papers that investigate an empirical relationship of some kind between >=1 identifiable independent variable and >= 1 identifiable dependent variable, where the independent variables are characteristics of software artifacts and the dependent variables are defects during maintenance.
Exclusion Criteria	• Papers that present an empirical strategy different from a controlled experiment (e.g., a case study) • Papers not subject to peer review • Papers that do not provide a *p-value*

One of the reasons to include only controlled experiments, excluding repository analyses, is that **causality can be found only by holding constant the object under study**; i.e., the software that forms the object of study must differ from one trial to another only in the value(s) of the independent variable(s). Otherwise, when different values for the independent variables are present in software objects that are also different in other ways, then we cannot isolate the effect of the variables we are studying. Suppose two software objects have differing values for a complexity metric and different numbers of defects. If the two software objects are the same, then we can conclude that there is a causal relationship between that complexity metric and defects. But if the objects are different, then we can make no such conclusion as the relationship between the complexity metric and defects could be mediated by any number of other factors. A reliable way to create a consistent object for finding causal relationships is to clone the same object in two versions and manually modify one version by changing the independent variable. The two versions would then be subject to the same maintenance activity; the eventual difference in the dependent variable will exist only if the independent variable influences the dependent one. Study designs that don't do this are useful anyhow because using multiple different software objects allows more observations or the analysis of more independent variables.

We decided to accept all controlled experiments regardless of whether they hold the object constant or not. However, because studies holding the object constant provide more reliable results, in the analysis below we separate results based on this aspect. The selection procedure consists of two steps:

1) The entire set of 78 papers was divided among the authors.

2) To ensure consistency, the authors collaborated on the final selection of the papers to be included and the analysis and data extraction. Thus, the authors discussed with each other the proposed score and rationale, and they agreed on a classification.

B. Data Extraction and Synthesis

For each paper, we recorded the title, DOI, year, subject type, number of subjects, significance of results, research questions addressed, experimental object held constant, subject held constant, dependent variables, blocked variables, measured variables, and independent variables. Discrepancies during collaboration in the data extraction phase were resolved by a consensus between the authors. For each independent variable extracted, the associated statistical significance of the variable was recorded. The statistical significance was determined by the p-value provided in each paper.

IV. RESULTS

A. Overview of Studies

The sources that we queried provided us with a total of 76 papers[1] plus two papers that were added via snowballing. The complete results are available online[1]. Figure 2 shows the 78 papers returned from the query, broken down by their reason for rejection or validity in acceptance. Reasons for rejecting the paper include that the independent variable was not a software characteristic but rather the inspection technique used to find defects or that the empirical procedure was not a controlled experiment but rather a survey.Nine studies have been included in this MS [7] [2] [8] [9] [10] [11] [3] [12] [13]. Three studies hold the object constant. Two of these were highly related in that one [9] was a replication of the other [10]. Specifically, the aim of these two studies was to understand whether design patterns improve the maintainability and understandability of software versus a simpler alternative approach to design. The results showed that the use of some design patterns had a statistically significant effect on maintainability, including defects. The third paper that held the experimental objects constant [11] was designed to analyze the design of self-adaptive systems for the purpose of evaluating internal adaptation mechanisms and external monitor-analysis-plan-execute (MAPE) loops with respect to their fault density, among other things.

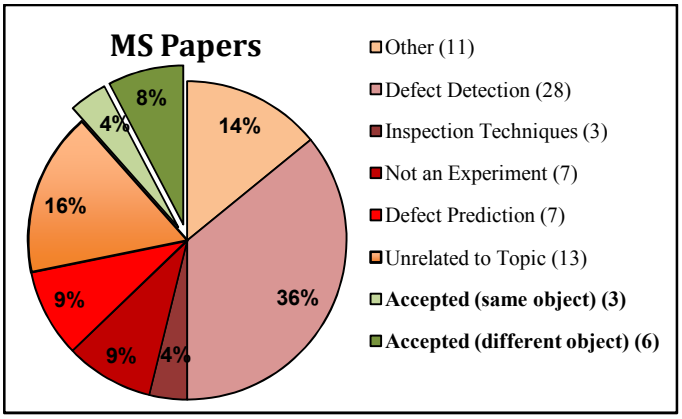

Figure 2: Breakdown of accepted and rejected papers.

B. Findings About Research Questions

1) RQ1: Do any software characteristics show a statistical impact on defects?

Within the nine accepted papers, we observed 42 different independent variables. Of these, 29 (58%) were found to be statistically significant and 21 (42%) were found not statistically significant. There was some overlap in the independent variables investigated; however the same independent variable did not show a statistical impact on defects in more than one paper. Thus we conclude that results are inconsistent because the same software characteristic did not always influence the number of defects.

2) RQ2: What is the distribution of the number of independent variables that have been considered across controlled experiments?

Figure 3 depicts a distribution of our analysis of

[1] http://goo.gl/mjLaJo

independent variables in the 9 (A-I) accepted papers. The number of independent variables explored within the controlled experiments ranges from 1 to 19. However, the number of independent variables is very low for studies holding the experimental object constant. Therefore, possible interactions among multiple independent variables may exist.

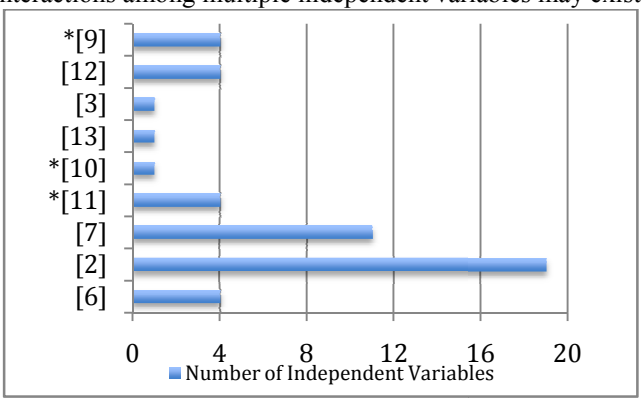

Figure 3: Distribution of the number of independent variables that have been considered across controlled experiments. An asterisk indicates a paper where the object in the experiment was held constant. The number in square brackets is the reference to the paper.

3) RQ3: Do any software characteristics have a causal relationship with defects in maintenance?

According to our analysis, no causal relationships were determined between software characteristics and number of defects during maintenance. Specifically, although several (29) independent variables were found to be statistically significant, 1) the same independent variable did not show a statistical impact on defects in more than one paper, and 2) each paper only looks at a subset of the independent variables that could affect defects in maintenance, and so no set of characteristics can conclusively be determined to have a causal relationship.

V. Conclusion

The goal of this study is to find and understand the software characteristics that play a role in producing maintainable code by developing a knowledge base about causal relationships between observable code characteristics and maintenance problems. Our research shows that in most of the cases, different objects are used in the same experiment and hence results may be driven by external variables. Moreover, the controlled experiments performed by holding the experimental objects constant only tested a few independent variables, and therefore could not confirm causality. Finally, the same independent variable did not show a statistical impact on defects in more than one paper. In conclusion, our study points to weak evidence on software factors causing defects during maintenance.

Stronger evidence can be developed by leveraging past correlational results via more controlled experiments that address multiple independent variables and hold the software objects constant. For instance we plan to perform multiple experiments testing the impact of, and interaction among complexity metrics during maintenance. Specifically, we plan to start investigating method size and class size, i.e., two complexity metrics analyzed by not holding the experimental object constant and resulting statistically significant to number of defects in maintenance [2]. To do so, we plan to create different experimental objects, by manually modifying the same code of an open-source software project, allowing for all combinations of these two complexity metrics (i.e., low-low, low-high, high-low, high-high). The number of defects, and hence the statistical significance of the impact of these variables on them, will be determined by subjecting all four objects to the same extensive maintenance activity. Results will help practitioners in understanding if and how much the size of their classes and their methods make their system more error prone.

We also plan to investigate casual factors originating from contexts, development processes and organizational aspects. Finally, even if exactly the same object is given as treatment to the subjects of the study, there might be confounding factors that dwarf the design factors like developers' skill and experience. Therefore, we will use fractional designs to properly explore at least some of the possible interactions.

References

[1] "IEEE Standard Glossary of Software Engineering Terminology." pp. 1–84, 1990.

[2] S. C. Misra, "Modeling design/Coding factors that drive maintainability of software systems," *Softw. Qual. J.*, vol. 13, pp. 297–320, 2005.

[3] D. Chatterji, J. C. Carver, N. A. Kraft, and J. Harder, "Effects of cloned code on software maintainability: A replicated developer study," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2013, pp. 112–121.

[4] F. P. Brooks, "No Silver Bullet Essence and Accidents of Software Engineering," *Computer (Long. Beach. Calif).*, vol. 20, no. 4, pp. 10–19, Apr. 1987.

[5] R. Eisenberg, "Management of Technical Debt: A Lockheed Martin Experience Report," *Fifth International Workshop on Managing Technical Debt*, 2013. http://goo.gl/oyZy3J. [Accessed: 23-Jun-2015].

[6] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," *Engineering*, vol. 2, p. 1051, 2007.

[7] F. Rahman and P. Devanbu, "Ownership, experience and defects," in *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, 2011, p. 491.

[8] Y. Takahashi M.; Kamayachi, ""An empirical study of a model for program error prediction," *Softw. Eng. IEEE Trans.*, vol. 15, pp. 82–86, 1989.

[9] A. Nanthaamornphong and J. C. Carver, "Design patterns in software maintenance: An experiment replication at university of Alabama," in *Proceedings - 2011 2nd International Workshop on Replication in Empirical Software Engineering Research, RESER 2011*, 2012, pp. 15–24.

[10] L. Prechelt, B. Unger, W. F. Tichy, P. Brössler, and L. G. Votta, "A controlled experiment in maintenance comparing design patterns to simpler solutions," *IEEE Trans. Softw. Eng.*, vol. 27, pp. 1134–1144, 2001.

[11] D. Weyns, M. Usman Iftikhar, and J. Soderlund, "Do external feedback loops improve the design of self-adaptive systems? A controlled experiment," in *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2013, pp. 3–12.

[12] A. . Gosain, S. . Nagpal, and S. . Sabharwal, "Validating dimension hierarchy metrics for the understandability of multidimensional models for data warehouse," *IET Softw.*, vol. 7, pp. 93–103, 2013.

[13] J. Harder and R. Tiarks, "A controlled experiment on software clones," in *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, 2012, pp. 219–228.