Surveying the Factors that Influence Maintainability

Research Design

Wiebe Hordijk[*]
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
www.cs.utwente.nl/∼hordijkwtb
hordijkwtb@cs.utwente.nl

Roel Wieringa
Faculty of Electrical Engineering
Mathematics and Computer Science
University of Twente
www.cs.utwente.nl/∼roelw
roelw@cs.utwente.nl

ABSTRACT

We want to explore and analyse design decisions that influence maintainability of software. Software maintainability is important because the effort expended on changes and fixes in software is a major cost driver. We take an empirical, qualitative approach, by investigating cases where a change has cost more or less than comparable changes, and analysing the causes for those differences. We will use this analysis of causes as input to following research in which the individual contributions of a selection of those causes will be quantitatively analysed.

Categories and Subject Descriptors

D.2.11 [**Software**]: Software Architectures; D.2.9 [**Software Engineering**]: Management—*time estimation, productivity*; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*restructuring, reverse engineering, and re-engineering*; D.2.8 [**Software Engineering**]: Metrics—*product metrics*

General Terms

Design, Measurement

Keywords

Software maintainability, software design, case study

1. INTRODUCTION

This research is part of a research project that aims at identifying and validating relations between design decisions and software quality. In this part of the project, the quality attribute *maintainability* has been chosen as the prime attribute of interest. To establish a solid theory of maintainability, the factors that are thought to influence maintainability (also known as cost drivers) must be explored first, and later validated.

The system of which *maintainability* is an attribute, is the combination of the software system and the maintenance team, where the team is the (part of the) organisation that performs changes to the software. Maintainability cannot be seen as an attribute of the software system alone, because it depends a great deal on who maintains it: a team that has a lot of experience with a particular system will maintain it more easily. Both the software and the team have internal attributes that influence maintainability, for example, structural complexity of the software and skill of the team members. We want to survey the factors that lead to low or high maintainability.

As an indicator of maintainability we choose *change effort*. Change effort is an external attribute of a change. Change effort is the total of analysis and programming effort expended on one particular change. Change effort does not include related activities, such as documentation, communication or deployment. We do not need a system-level maintainability measure, as the individual change efforts are the object of our study.

A change request can be due to a failure, changing requirements, prevention or any other reason. The activities by the maintenance team include actually performing the change, but also documenting, testing, and reporting, depending on the maintenance procedures. When a system is changed so extensively that a new team is formed to implement the changes, that is not regarded as a change. Such a situation is more like a new system being developed.

Based on our own experience and preliminary literature study, we think that the factors that influence maintainability can be categorised in the following way (assembled and adapted from [1], [2], [4], [6], [7], [8] and [9]). This categorisation is used as an aid to direct our exploration and to keep it within reasonable bounds. When during the research we find that categories should be added or changed, that will be an outcome of the research.

- Software product properties
  - Specification-level properties, e.g. correctness or functional size
  - Design-level properties, e.g. modularity, coupling

[*]Sponsored by Ordina (www.ordina.nl)

    - Code-level properties, e.g. code size, complexity, maturity, decay, duplication
- Maintenance process properties
    - Frequency of changes
    - Maintenance procedures
    - Testing process properties, e.g. test coverage
- Resource properties
    - Team properties, e.g. activity rate, communication structure, personnel turnover
    - Team member properties, e.g. skill level, familiarity with the system
    - Maintenance infrastructure, e.g. development environment, tools
- Change properties
    - Change functional properties, e.g. functional size, change type, functional complexity, performance sensitivity
    - Change implementation properties, e.g. change size, change span, fault potential

In our research, we treat design decisions as independent variables and change effort as the dependent variable. We assume the software product properties as intervening variables, linking the design decisions to change effort. The other factors are either control variables if we can isolate them, or extraneous variables if we cannot. This is illustrated in figure 1.

To be able to analyse the software product properties as intervening variables, we will need to make them measurable in some way. There are several approaches to measurement of these properties. One notably comprehensive set of instruments is provided by MITRE corporation [4]. We currently evaluate these instruments.

The change property *functional size* deserves special attention. We consider it logical, and not a sign of bad design or inefficient procedures, that larger changes take more effort. The functional size of changes, however, is not easy to define. It is a very different measure from the technical size, which can be expressed for example in lines of code. The functional size is the size of the change as seen from the user's viewpoint. This is important because it is only logical that bigger changes need more effort. The maintainability we're interested in, is the effort per unit volume of functional change size. To control this variable, we use a categorisation of functionally similar changes, where changes are only compared within their own category.

2. GOAL

Our goal is to investigate which design decisions influence maintenance effort for software systems. Our scope is limited to custom-built administrative enterprise applications.

Our first sub-goal is to explore which factors *could* influence software maintainability. We need an as complete as possible overview of these factors, to enable isolation of individual factors in the validation part of the research. The exploration has been successful if we have a model that at least covers the categories mentioned in the introduction, and that we can use to base further research on.

Our second sub-goal is to validate whether design decisions found in the exploration really have an effect on software maintainability.

3. RESEARCH DESIGN

To achieve our first goal we will review the literature for already investigated factors that influence maintainability, and visualise these in a cause-effect graph. We will expand this graph with factors from our own experience and maintenance personnel's opinions about what factors affect maintainability. In this graph, sources and evidence for the hypothesised cause-effect relations will be visualised.

To achieve our second goal we will validate the effects that design decisions have on maintainability. In each validation, we choose one particular design decision as the independent variable, and the maintainability of the system is the dependent variable. Design decisions are choices between finite numbers of options; they have a nominal scale and can be measured by inspecting the software product. We operationalise maintainability as the change effort of individual changes. Numerous other factors are either intervening or extraneous variables, which is why the exploratory part of this research is important: since there are long cause-effect chains between design decisions and maintainability, and there are many extraneous variables, having a good map of the factors involved is essential for isolating the variables under investigation.

We will validate the effects that design decisions have on maintainability by looking for cases where a difference in options chosen for a particular design decision can explain a difference in change efforts. This can be shown by finding pairs of cases where for a certain decision, all else remaining equal, in one case one option is chosen and in the other case the other, and then measuring the change effort in both cases. This is of course an idealised picture of the research design; in practice, more factors will be different between each two cases, so more than two cases will be needed to single out one factor. Some thoughts on isolating individual factors can be given. Team factors, for instance, can be isolated by picking cases where the same team maintains several systems. Some other factors can be isolated by taking cases from different parts of the same system, or the same system at different moments in time. This leads to the case study design depicted in figure 2.

The case study design we are currently testing in a pilot case study is a multiple-case embedded design [11], where the top-level units are systems and the embedded units are changes to those systems. We compare changes to the same system using contrasting logic, and between the systems we use replication logic, as shown in figure 2. The strategy for analysing individual cases is pattern matching: from the logical model given by the theoretical framework, we derive expected patterns in the data for the hypothesis and the rival explanations, and then we show whether the actual data matches the hypothesis pattern or any of the rival explanation patterns.

4. SAMPLING

We will select a sample of cases, based on the following criteria:

- Whether they have collected data about maintenance effort;

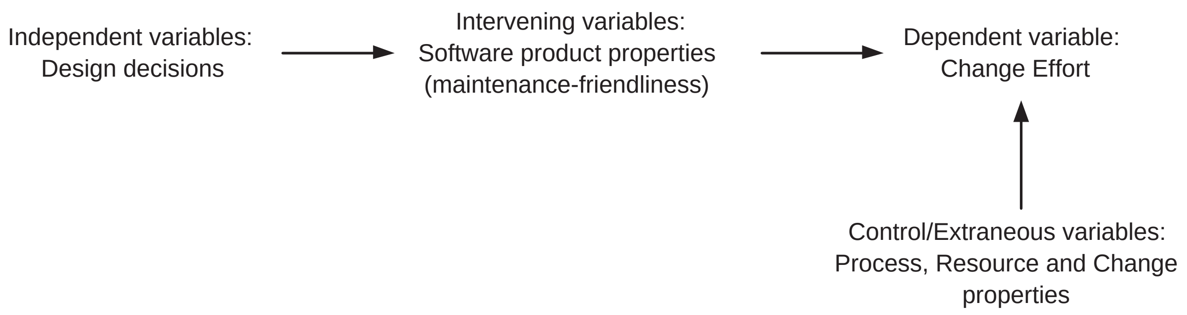
Independent variables:
Design decisions
Intervening variables:
Software product properties
(maintenance-friendliness)
Dependent variable:
Change Effort
Control/Extraneous variables:
Process, Resource and Change properties

Figure 1: Overview of the relations between the kinds of variables

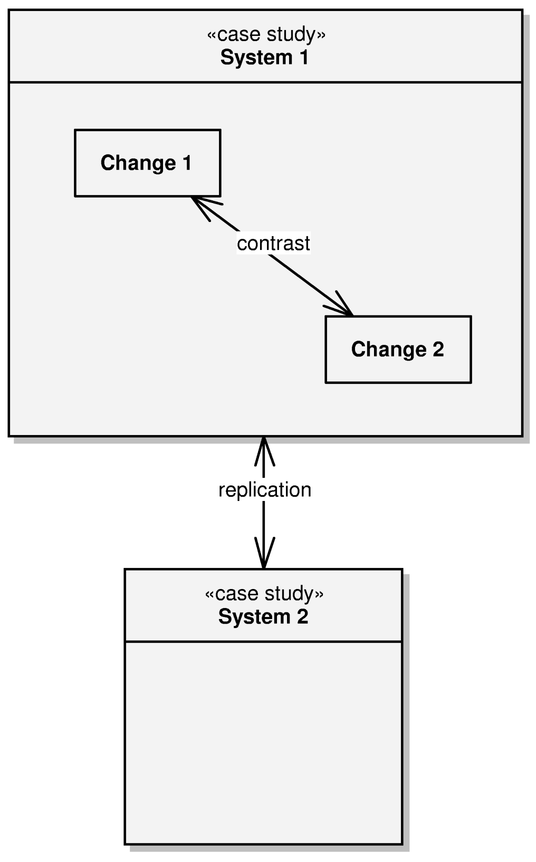

Figure 2: Overview of the case study design

- Whether enough documentation and people are available to investigate the causes of maintenance effort differences;
- Whether the systems themselves are representative for the class of systems in our scope;
- We will choose cases that have occurred in recent history, to facilitate data collection.

5. RATIONALE

The reason we choose case study research as our research method is that case study research is particularly suited for research in which the number of variables is greater than the number of data points that can feasibly be sampled [11]. Since maintainability is influenced by many factors, this is certainly the case for our research. Also, case studies are appropriate for descriptive research like ours, in which the causes of phenomena are investigated.

We choose cases from history to facilitate data collection. We prefer as recent cases as possible, to improve the relevance of the results (as technological advancements make results from older cases obsolete) and to improve the accuracy of the data (as we partly need to rely on the memories of people involved).

We first planned to focus on maintenance outliers, because we thought that those would yield more qualitative information than normal changes. We figured that a practitioner when asked why change X took an average change effort might not know what to answer, whereas the same interviewee would give lots of valuable information when asked why change Y took ten times as much. We have deviated from that approach because of the foreseeable difficulties in generalising from outliers to normal cases. When our chosen approach yields too little results, we can still decide to use the outlier approach.

We operationalise maintainability as the change effort of individual changes. We choose this fine level of granularity instead of a coarser measure like average change effort, to benefit from the additional information that can be retrieved from differences between different parts of the same system.

We choose a rather narrow definition for change effort, excluding all activities except analysis and coding. This is because for many of the excluded activities, the team can choose how much effort is expended on them. Take documentation effort as an example. The more documentation effort, the higher the quality of the documentation (we think). The higher the quality of the documentation, the more maintainable the system will probably be. So if we would measure documentation effort as part of change effort, we would contaminate our dependent variable with an independent variable.

We rank changes into similarity classes, instead of using a measure for functional size of changes based on a ratio scale. This is because we expect considerable difficulties in choosing such a ratio scale. Some researchers [1] report success with using function points for measuring change size in small maintenance projects, while others [10] report low fidelity of function point measures for change size. We do not need a ratio scale for change size in this research (although we may need one in further research), so we avoid such difficulties for now by choosing a nominal scale. We still risk interaction between the categorisation and change effort, i.e. different categorisations of a set of changes may yield different influences of factors. We don't see a solution to this weakness, but we reduce the problem by being transparent about the categorisation.

6. ACKNOWLEDGEMENTS

We thank prof. dr. Klaas van den Berg for valuable insights about measurement of software system properties. We also thank dr. Denis Verhoef and dr. Jan-Willem Hubbers for valuable feedback on early drafts of this research design.

We thank the anonymous reviewers for pointing out unclear parts of the research design and for recommending some very interesting previous works.

7. REFERENCES

[1] Y. Ahn, J. Suh, S. Kim, and H. Kim. The software maintenance project effort estimation model based on function points. *Journal of Software Maintenance*, 15(2):71–85, 2003.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 2nd edition, 2003.

[3] M. A. Chaumun, H. Kabaili, R. K. Keller, F. Lustman, and G. Saint-Denis. Design properties and object-oriented software changeability. In *CSMR '00: Proceedings of the Conference on Software Maintenance and Reengineering*, page 45. IEEE Computer Society, 2000.

[4] M.-A. Côté, W. Suryn, C. Y. Laporte, and R. A. Martin. The evolution path for industrial software quality evaluation methods applying ISO/IEC 9126:2001 quality model: Example of MITRE's SQAE method. *Software Quality Journal*, 13:17–30, 2005.

[5] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus. Does code decay? Assessing the evidence from change management data. *IEEE Trans. Softw. Eng.*, 27(1):1–12, 2001.

[6] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing Co., 1998.

[7] T. L. Graves and A. Mockus. Inferring change effort from configuration management databases. In *METRICS '98: Proceedings of the 5th International Symposium on Software Metrics*, page 267. IEEE Computer Society, 1998.

[8] M. Lehman. Laws of Software Evolution Revisited. In C. Montangero, editor, *Software Process Technology (EWSPT 96)*, volume 1149 of *Lecture Notes in Computer Science*, pages 108–124, Nancy, France, 1996. Springer-Verlag, Berlin.

[9] A. Monden, D. Nakae, T. Kamiya, S. Sato, and K. Matsumoto. Software quality analysis by code clones in industrial legacy software. In *METRICS '02: Proceedings of the 8th International Symposium on Software Metrics*, 2002.

[10] F. Niessink and H. van Vliet. Predicting maintenance effort with function points. In *ICSM '97: Proceedings of the International Conference on Software Maintenance*, pages 32–39. IEEE Computer Society, 1997.

[11] R. K. Yin. *Case study research: design and methods.* Sage Publications, 3rd edition, 2003.