A multi-method approach for evaluating software maintainability and comprehensibility

Aiko Fallas Yamashita

Software Engineering Department
Simula Research Laboratory, Box 134, 1325 Lysaker, Norway
aiko@simula.no

Abstract. The maintainability of software is a critical determinant of software costs, yet a very difficult attribute to evaluate. Insofar, no unified theory is available regarding how to use design attributes of a software system (represented either by structural measures[1] , structural symptoms[2] or a combination of both) to evaluate accurately the maintainability of a system. Construct validity of software measures, as well as realistic maintainability models/evaluation techniques, are still subject of a latent discussion in the software engineering community.

The diversity of contexts and situations makes it difficult to generalize findings from individual studies, and the inherent complexity of software maintenance (encompassing technological, human dynamics and cognition aspects) make the assessment of maintainability a difficult task. In those regards, we have a similar viewpoint as Pizka & Deißenböck [7] as they state: *"maintainability is not solely a property of a system but touches three different dimensions: a) The skills of the organization performing software maintenance, b) Technical properties of the system under consideration and c) Requirements engineering"*.

Comprehension approaches such as opportunistic comprehension by Letovsky [4] and opportunistic recognition of program plans (Soloway & Ehrlich, 1984) [9] assume that a knowledge base is composed by the programmer expertise and their background knowledge. Work by O'Brian [6] proposes that understanding large commercial software systems is driven by an information requirement. This suggests that one of the most important maintenance activities (program comprehension) is driven by the programmers experience and the information need (this last one we conjecture, is driven by the nature of maintenance tasks).

Therefore, our approach includes a more holistic approach to maintainability evaluations than that taken in most earlier studies and combine not only design attributes but incorporate also maintenance tasks and developers skills into the analysis. A study performed by Anda [1] has suggested that a combination of expert judgment and analysis of software design attributes is likely to be a viable evaluation approach, since they address different attributes and dimensions of a system. We also conjecture that expert judgment might incorporate the analysis of contextual and cognitive factors into the maintainability evaluation, enabling more realistic and flexible models that can be used in combination with the already existing software design attributes analysis.

This PhD work builds upon this premise and intends to undertake several follow-up studies from the findings in [i], focusing on three main areas: (a) Determine the impact of different software design attributes on the ease/difficulty of different maintenance tasks (b) Establish a framework for combining software design attributes and expert judgment in maintainability evaluations and (c) Determine the adequacy/qualification of expert knowledge.

Keywords: POP-I.B. Barriers to Programming; POP-II.B. Program Comprehension; POP-II.B. Maintenance; POP-IV.A. Object Oriented Design; POP-V.B. Case Studies.

[1] By structural measures we refer to structural properties of software that can be quantified such as size, coupling and cohesion and can be used as indicators of maintainability. Anda [1] indicated that research on maintainability has focused mostly on the maintainability of classes or clusters of classes of individual systems, while the maintainability of complete software systems has received relatively little attention.

[2] By structural symptoms we refer to the term code smells first coined by Kent Beck and Martin Fowler. Code smells are surface indicators in the program design, which usually correspond to a deeper problem in the system. It has been proposed in order to provide a guidance for recognizing situations where refactoring might be needed. Recently, the work by Marinescu has help to define strategies for detecting of some of these indicators, enabling their automated detection/measurement in a system.

1 Determining the impact of software design on maintenance tasks

Relying only on structural measures does not offer clear guidance to developers about how to improve maintainability of a system. Our motivation for integrating structural symptoms into the analysis is that for each of these symptoms, re-design strategies (e.g., refactoring, use of design patterns and design heuristics) can be used to improve the software. However the cost of implementing such strategies must be related to the cost of having such flaws in the system. To our knowledge, earlier studies of structural symptoms have not analyzed yet their effects on maintainability and program comprehension.

Ehrlich found that expert programmers performed better on plan-like programs, than on unplan-like programs. They suggested that this was due to their experience with programming, and their tendency to recognize familiar rules of the programming discourse. On unplan-like programs, Soloway noticed that experts' performance deteriorated, as they seemed confused by the rule violations, and indeed their performance levels dropped to near the level of the novice programmers. Implications of unplan-like programs are high if they slow-down the performance of expert developers. Since structural symptoms can also be seen as anti-patterns or design principle violations, they could be used as indictors of unplan-like programs. Many comprehension models such as [9] and [11] are based on the notion of beacons or information cues, which are used by developers to seek, relate and collect information (Ko et. al, 2007) [3]. If the cues are misleading, the developers might be affected and loose time inspecting insignificant code. Early work on information cues has been used to improve the design of IDE tools, but virtually no work focuses on software design attributes. For instance, we might ask how different structural symptoms (e.g., inconsistent names combined with dead code) can affect the perception of the developer about relevant information (e.g., vocabulary problem). To which point could they be misleading? What is the impact on a real-life project? We would also like to know better how structural measures (such as complexity measures) affect the different activities involved during a comprehension process.

In order to address these questions, we intend to conduct a multiple case study. In our case study context, we have the exceptional case of four systems developed from a common requirements specification (resulting in a implementation with minor differences in their functionality) and considerable dissimilarities in their design, size and complexity. The applications manage information related to scientific studies, and provide facilities for reporting. The systems were used for a period of time but currently are out of use because their interfaces are no longer compatible with the external systems from which they retrieved data. This gives us the chance to study a realistic case of an adaptive maintenance process. Having these four systems enables us to perform a comparative case study where we can ask developers to perform identical maintenance tasks on the different systems, and observe the differences among the processes. Other studies have performed comparative case studies involving systems with different functionalities and contexts, which complicates the generalization of results. The conditions of our study enable a better level of control over the maintenance tasks and the domain area, thus facilitating cross-comparison and theoretical replication without loosing realism.

Considering the above conditions and rationale, we propose the use of a multiple case study with control of the software functionality variable of the cases (we term this research method: "controlled, multiple case study"). We aim to explore the existence of clearly distinguishable design attributes that may determine meaningful differences on the ease/difficulty of maintenance tasks among systems with different design characteristics.

The use of Pattern Matching (PM) is suggested for assuring internal validity (such as cause-effect relationship) as well as for ensuring that inferences are correct (Yin, 1994) [13]. Trochim [10] has used PM for outcome program evaluations or causal assessments, from which we imply that we can also use it for studying the effects of software design in a complex setting such as a maintenance project.PM involves an attempt to link two patterns where one is a theoretical pattern and the other is an observed or operational one. The inferential task involves the attempt

to relate, link or match these two patterns. To the extent that the patterns match, one can conclude that the theory and any other theories, which might predict the same observed pattern receive support. We plan to draw several patterns supporting our study proposition (an "effect pattern" on maintenance effort and program comprehension) and also patterns supporting rival theories or explanations. Rival theories (direct rivals or commingled) can express alternative assumptions about comprehensibility and effort required on different systems in relation to each other. A rival explanation could be that factors not represented by design attributes may have a stronger impact on the maintainability than the design. One potential such factor is logical connections among the software components. Rival explanations are an important aspect of case studies, since they may help us to refine our data collection methods. For instance, by focusing our attention on possible "other influences" we will avoid being accused of favouring the data collection towards our original hypothesis (Yin, 1994) [13].

We intend to adapt the protocol used by Ko et al. [3] in order to identify the different comprehension activities (seek, relate and collect information) but our focus will be on cues related to software structural measures and symptoms instead of GUI or visual elements. The different comprehension activities and situations where the developer has found relevant information will be detected by using several methods such as: think-aloud protocol during the planning stage of the task and daily semi-structured interviews with the developers to follow their progress during the implementation of the maintenance tasks. We intend also to record the developers' maintenance activities (edit, search, navigate, etc) through an Eclipse plug-in so we can observe the maintenance and comprehension activities in detail. Using several sources for detecting the comprehension activities will allow better certainty in the results (triangulation). By using comprehension models, we intend to refine our study protocol and also formulate steering questions for the semi-structured interviews. We will also focus on the different types of information (Jarvelin & Repo, 1983) [2] that the developers might search, which we expect will be dependant of the nature of the different maintenance tasks.

2 Establish a framework for combining software design attributes and expert judgment in maintainability evaluations

Constructs for representing maintainability (since as we said maintainability is a context dependent attribute) should be to some extent goal-driven. This means that measures could represent different constructs, which may have different weights or priorities depending of the domain and nature of the project. We conjecture that most of the issues with construct validity of software quality measures (suggested by standards such as ISO) are due to its static nature, which don' t address the real needs of the project or goals of the organization. We need a flexible framework that can help us to build and ensure validity of the maintainability constructs in a wide range of contexts. We believe that expert judgment might incorporate contextual and cognitive factors to the analysis, which could provide more accurate representations of maintainability. For instance, we have tried the conceptualization task suggested by Trochim[10] to derive conceptual maps using structural measures and structural symptoms relevant to the 4 systems under study. The result was a sort of tailored "maintainability ontology" on the form of a two-dimensional point map composed by measures, which were clustered into constructs representing different attributes related to maintainability (this process has some similar features to factor analysis). We have used one expert programmer (with more than 10 years of experience) and academicians to perform the grouping, all with good knowledge of the systems. An interesting result was that the clustering of measures done by the researchers differed considerably from the one made by the expert. Where the grouping of structural symptoms performed by the researchers was based on a taxonomy similar to the one proposed by Mantyla [5] (following a design heuristics ontology), the one performed by the expert related more to difficulty and uncertainty levels of each of the symptoms as well as the potential severity they represent (following more of a

risk analysis view point). This initial experience indicates that such conceptualization could be a viable way to formalize and represent the knowledge of experts thus supporting theory building (theories come from formal tradition of theorizing, "hunches" or combination of both) in the maintainability area. Conceptualization or concept mapping could be a flexible choice to be used in order to build goal-oriented constructs (by using the already existent measures but using experts to conjecture their value and interpret them) more fit to the specific context of analysis. Trochim has stated that: *"Group concept mapping is consistent with the growing interest in the role of theory in the thinking of critical multiplism (Shadish, Cook, & Houts, 1986) [12] which emphasizes the role of theory in selecting and guiding the analysis of multiple operationalizations"*. Conceptual maps are useful as theoretical patterns that can be used in pattern-oriented research strategies. Richter [8] has commented on the need for these types of strategies in order to bridge the gap between theory and practice in complex social settings. The ideas underlying this approach are rooted in theory-based evaluation, realistic evaluation and explanatory case studies (Richter, 2004). Concept mapping (by using more or less established measures for maintainability), have tentatively been used to build constructs, this was in our case part of our research method, but considering our recent experiences, we also propose it as a method to be used in practice.

We propose PM as part of our research method to evaluate the identified maintainability constructs. We believe PM could aid improving construct validity in software quality indicators, especially for program maintainability and comprehensibility. PM started from the need of a "nomological network" in the disciplines of psychology (i.e., psychometrics) and social sciences (i.e., program evaluation) and has proven effective for addressing construct validity issues in both domains.

3 Determine the adequacy/qualification of expert knowledge

We would also like to explore means for assessing the accuracy of expert judgement (i.e., to what extent do experts with different backgrounds agree?) and determine guidelines for determining expert qualification (in which conditions it is feasible to use expert judgement and what qualifications are needed for such assessments?)

4 Plan for PhD

We will conduct a controlled-multiple case study, the upcoming autumn in order to address the points described in (a) and (b), so we expect to get enough data to address most of the questions formulated in these sections. (c) Will be addressed through several observational studies in collaboration with industrial practitioners.

5 Summary

- A holistic view on the evaluation of maintainability is proposed, which contemplates the software design attributes and the context of the software (developers and tasks).
- We plan to use a combination between structural analysis and expert judgment for deriving more realistic and well-contextualized assessments of maintainability.
- We want to determine effects of different design attributes on different maintenance tasks, from program comprehension perspective and other aspects of the maintenance process.
- In order to determine the effects, we propose the usage of pattern-oriented research strategies, in order to depict more clearly our assumptions on the effects of design attributes, and provide stronger support to internal and construct validity of the findings.
- We consider that conceptual mapping could be a good choice to operationalize expert knowledge into a maintainability evaluation framework.

6 Challenges for the work

- Relatively little work has been done in using pattern matching as a research method in Software Engineering (we need experiences from other fields such as psychology and social sciences in regards to our approach).
- We need to understand better and formulate the theoretical patterns of possible implications of structural symptoms on program comprehension and we might overlook important features of comprehension processes during this process.
- Description of rival theories in the form of theoretical patterns is still on its initial stage.

References

1. Bente Cecilie Dahlum Anda. Assessing software system maintainability using structural measures and expert assessments. In Gerardo Canfora and Ladan Tahvildari, editors, *the 23rd International Conference on Software Maintenance (ICSM 2007, the paper received Best Paper Award)*, pages 204–213. IEEE, 2007.
2. Jarvelin K. and Repo A. On the impacts of modern information technology on information needs and seeking: A framework. *In H. J. Dietschmann (Ed.), Representation and exchange of knowledge as a basis of information processes*, pages 207–230, 1983.
3. A.J. Ko, B.A. Myers, M.J. Coblenz, and H.H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *Software Engineering, IEEE Transactions on*, 32(12):971–987, Dec. 2006.
4. Stanley Letovsky. Cognitive processes in program comprehension. In *Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers*, pages 58–79, Norwood, NJ, USA, 1986. Ablex Publishing Corp.
5. M. Mantyla, J. Vanhanen, and C. Lassenius. A taxonomy and an initial empirical study of bad smells in code. *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, pages 381–384, Sept. 2003.
6. Michael P. O'Brien. Software comprehension - a review and research direction. Technical Report UL-CSIS-03-3, University of Limerick, November 2003.
7. Markus Pizka and Florian Deissenboeck. How to effectively define and measure maintainability. In Ton Dekkers, editor, *SMEF 2007 - 4th Software Measurement European Forum*, Rome, Italy, may 2007.
8. Christoph Richter and Heidrun Allert. Outline of a pattern-oriented research strategy for complex learning scenarios. In *ICLS '04: Proceedings of the 6th international conference on Learning sciences*, pages 427–434. International Society of the Learning Sciences, 2004.
9. Elliot Soloway and Kate Ehrlich. Empirical studies of programming knowledge. *IEEE Trans. Software Eng.*, 10(5):595–609, 1984.
10. William M. K. Trochim. Outcome pattern matching and program theory. *Evaluation and Program Planning*, 12(4):355–366, 1989.
11. A. von Mayrhauser and A. M. Vans. Comprehension processes during large scale maintenance. In *Proceedings of the 16th international conference on Software engineering*, pages 39–48, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.
12. T.D. Cook W. R. Shadish and A.C. Houts. Quasi-experimentation in a critical multiplist mode. *Advances in quasi-experimentul design and analysis. New Directions for Program Evaluation*, pages 29–46, 1986.
13. R. K Yin. *Case Study Research: Design and Methods.* Thousand Oaks, CA: Sage, 1994.