Modifiability Measurement from a Task Complexity Perspective: A Feasibility Study

Lulu He
Dept. of Computer Science and Engineering
Mississippi State University
Mississippi State, MS 39759
lh221@cse.msstate.edu

Jeffrey Carver
Department of Computer Science
University of Alabama
Tuscaloosa, AL 35487
carver@cs.ua.edu

Abstract

Despite the critical role of software modifiability, it has no universally accepted measurement model. Measuring modifiability in terms of maintenance effort is problematic because it confounds modifiability with the ability of individual maintainers. In this paper, we apply Wood's task complexity model to propose a general analytical model that describes the characteristics of maintenance tasks and the analytical dimensions of modifiability independent of the individual maintainers. The results of a case study demonstrate the construct validity of the model.

1. Introduction

Developing software that is easy to change is difficult. Due to constantly evolving requirements and hardware, most software is modified many times after its first release. Recent studies report that more than 90% of software costs are caused by maintenance and evolution [1]. Therefore, stakeholders expect a system to be designed so that it can be changed quickly and economically. This quality is referred as modifiability.

Modifiability is the ease of changing a system or component in response to a change request [2]. Other terms such as maintainability, changeability, and flexibility are often used to describe the same concept. Despite the critical role of software modifiability, there is no universally accepted measurement model. Sometimes modifiability is measured as maintenance performance, in terms of maintenance effort [3, 4], the number of faults introduced [5], or "perceived modifiability" – subjective opinion about the ease of changing software [6]. Sometimes modifiability is measured by a model of internal quality attributes (e.g. structural design properties) [7]. Neither performance measures nor internal quality models have sufficient construct validity for modifiability measurement. The former confounds modifiability with individual attributes, substituting a dependent variable (the outcome of the process) for an independent variable (inputs to the process). The latter ignores the effect of the change request, which is specified by definition. In this paper we propose a modifiability measurement model to address these problems.

2. Background

If software maintenance is treated as an information-processing task, then the theory of tasks analysis can be applied. Wood proposed the task complexity model to address the lack of an adequate theoretical model for describing task variation in studies of human behavior [8]. This variation makes it difficult or impossible to integrate evidence of task effects from different studies. Wood adopted a theoretical approach and synthesized previous analytical frameworks into a general model of tasks with three essential components: **products**, **acts**, and **info cues** [8]. The products are the output of the task; the acts and info cues are the input to the task.

Products: "*entities created or produced by behaviors, which can be observed and described independently of the behaviors or acts that produce them*" [8].

Acts: "*the patterns of behaviors with some identifiable purpose or direction*" [8]

Info cues: "*pieces of information about the attributes of stimulus objects upon which an individual can base the judgments he or she is required to make during the performance of a task*" [8]

Three analytical dimensions describe task complexity: **component**, **coordinate** and **dynamic**. Total complexity is determined by all three dimensions.

Component Complexity: "*function of the number of distinct acts that need to be executed in the*

978-1-4244-4841-8/09/$25.00 ©2009 IEEE

performance of the task and the number of distinct info cues that must be processed in the performance of those acts" [8].

Coordinative Complexity: "*nature of relationships between task inputs and task products*" [8]. The form, strength, and sequencing of the relationships are all considered to be aspects of coordinative complexity.

Dynamic Complexity: "*changes in the states of the world which have an effect on the relationships between tasks and products*" [8].

The components and dimensions of complexity in Wood's model describe stable task properties that can be specified independently of the task performers.

3. A Model of Software Modifiability

3.1. Model Constructs

Wood's model provides a general approach to task complexity analysis. Task complexity characterizes the difference among task inputs and the relationship between task inputs and outputs. Software maintenance can be viewed as an information-processing task in which maintainers perceive, interpret and manipulate info cues (task inputs) and the relationships among them to identify task outcomes [5]. The info cues and the acts required to process these cues set upper limits on the knowledge, skills and resources required to successfully complete the task. Therefore, the required acts, info cues, and the relationship among them help identify the difficulty of the maintenance task. It is natural to draw parallels between task complexity and software modifiability. Wood's model allows us to describe the properties and difficulty of the maintenance task independent of the maintainers. This independence differentiates our model from existing performance measures. By mapping Wood's model to software maintenance, we derived the model of maintenance tasks and modifiability shown in Table 1.

Acts are defined as actions dealing with code. Because our definition of modifiability is the ease of changing a software system, we focus on the actions required to implement a change, i.e. adding, modifying or deleting code. While the change process does involve actions other than "coding", like, "reading" and "thinking", those cognitive activities are difficult, if not impossible, to identify from the maintenance tasks. The coding actions, by contrast, are more "physical" and thus easier to specify and identify.

Info cues (ic) are defined as the information pieces from the change request or the software system because these are the two sources of difficulty.

Our model specifies modifiability in terms of task complexity. A more complex maintenance task leads to less modifiable software. In other words, a high value for component complexity or coordinate complexity indicates low modifiability.

Table 1 Mapping Wood's Model to Software Modifiability

Wood's Model [8]	Modifiability Model
Product	Changed Software System
Act	Different types of change actions, e.g. addition, deletion, or modification of code
Info Cues	Pieces of information required from software systems and change requests to perform the acts properly.
Component Complexity	A measure of the number of change actions, weighted by the amount of information cues processed by that action
Coordinate Complexity	A measure of the relationship (e.g. ordering) between different change actions
Dynamic Complexity	Not considered in current model.

3.2. Measurement Protocol

The proposed model is abstract enough to apply to a variety of maintenance environments. For validation, we need a measurement protocol to obtain the measurement values consistently and repeatably. Therefore, we specified a measurement process of identifying model components (acts, info cues) and calculating modifiability in different dimensions (component, coordinate).

Wood's model can specify task complexity independently of individual attributes because the analytical constructs are identified a priori from the task specification. In the case of software maintenance tasks, however, the change request and the system specify the requirements for the change, rather than how to make the change. The acts and info cues are not as evident in this context as they are in Wood's.

Instead of guessing the acts and info cues for a change request and system, we examine the actual change implementations by maintainers. Acts can be identified by determining whether code is added, modified, or deleted. The info cues are then limited to the information pieces referenced in the changed code. We are unable to track the cognitive activities of the maintainers. Thus we can only make inferences about the information processed by them based on what is actually documented, i.e. the changes made to the code. These info cues set the minimum requirement on the information needed to perform the change task. The maintainers must acquire and process these information pieces before they can reference them in

978-1-4244-4841-8/09/$25.00 ©2009 IEEE

the code. The only problem is that the acts and info cues identified may be affected by individual maintainers (e.g. programming style, expertise). Our hypothesis is that there is a subset of acts and info cues that are determined by the change request and software system, independent of individual maintainers. This hypothesis will be tested in model validation.

The info cues are extracted in units, i.e. Program Elements (PEs), including *attributes*, *local variables*, *parameters*, *methods*, *classes*, and *constants*. We refer methods as M_type PEs and the other PEs as O_type. While these PEs are specific to the Java language, similar PEs can be defined for other OO languages.

For the modifiability dimensions, we began with component complexity. Coordinate complexity will be examined in future. To measure component complexity, we need to specify how to measure a single info cue and how to calculate the overall complexity for all the info cues.

To measure the amount of information in a single info cue, we have initially chosen a simple size measure, assuming that the larger an information cue is, the more information it contains. We developed three size measures as follows (the size of an info cue ic is denoted as S(ic)):

- For info cues *ic* from change requests, S(*ic*)=1.
- For O_type *ic* (from the original code), if *ic* is not *Class* or instance of *Class* (e.g. int, String, etc.), then S(*ic*)=1.

S1: **LOC** (lines of code)

- For O_type *ic* if *ic* is an instance of *Class C*, **S1**(*ic*) = **S1**(*C*).
- For M_type *ic*, **S1**(*ic*) = **LOC**(*ic*).
- For *Class C*, **S1**(*C*) = **LOC**(*C*).

S2: **MImpl** (Method Implementation)

- For O_type *ic* if *ic* is an instance of *Class C*, **S2**(*ic*) = **S2**(*C*).
- For M_type *ic*, **S2**(*ic*) = ∑ (#Parameters, #Methods_invoked, #Attributes_referenced).
- For *Class C*, **S2**(*C*) = ∑(#Attributes, **S2**(Methods)).

S3: **MIntl** (Method Interface)

- For O_type *ic* if *ic* is an instance of *Class C*, **S3**(*ic*) = **S3**(*C*).
- For M_type *ic*, **S3**(*ic*) = ∑(#Parameters)
- For *Class C*, **S3**(*C*) = ∑(#Attributes, **S3**(Methods))

S1 measures the complexity of the info cues in terms of the lines of code. S2 and S3 measure methods in terms of the number of PEs referenced. The difference between S2 and S3 is that S2 examines the method (white box view) while S3 examines only the interface (black box view). The underlying assumption of S2 is that method usage requires knowledge of method implementation, while for S3, only the interface is needed to use the method.

In some cases, an ic is referenced more than once. It is unclear whether it is more difficult to process an ic the first time it is encountered than it is on subsequent encounters. We define an integration rule for each case: R assumes there is no difference in difficulty for subsequent observations of an ic while R' assumes that subsequent encounters of an ic are easier to process. Therefore, there is 1 extra unit of complexity added for each subsequent encounter rather than adding in the full size.

- **R**: $TS = N^* \mathbf{S}(ic)$
- **R'**: $TS = \mathbf{S}(ic) + (N\text{-}1)$

TS denotes the total size of all the ics and N denotes the number of total encounters. Combining each measurement approach (S1, S2, S3) with each integration approach (R and R') results in six measures M1, M1', M2, M2', M3, M3'. We tested our model for all six measures in the case study in section 4.

3.3 Model Evaluation

We demonstrate the construct validity of our model from the following three aspects:

Face Validity: the components in the modifiability model are determined by software system and change request only, independent of individual maintainers.

Concurrent Validity: the modifiability model can distinguish the difficulty of maintenance tasks.

Predictive Validity: the modifiability model can predict the maintenance performance of the same maintainers on different maintenance tasks

4. Case Study

Before conducting our own controlled experiment, we validated our model using the data from an experiment conducted by Arisholm, et al. [9]. The goal of the original experiment was to compare the changeability of two designs, i.e. Responsibility-Driven (RD) vs. Main Framework (MF). According to OO design principles, RD is the better design and thus should be more changeable. The concept of changeability is similar to modifiability as defined in our model. There were three change tasks. Each participant performed the changes on one of the two designs. The changeability of the designs was assessed based on change effort (time to finish the tasks), correctness (a subjective correctness score), and other measures. The results indicated that, contrary to the hypothesis, the good RD design required significantly more effort than the bad MF design and the RD design did not result in fewer errors than the MF design.

We posed five hypotheses to test three types of construct validity:

978-1-4244-4841-8/09/$25.00 ©2009 IEEE

H1 The *Acts* do not differ between subjects.
H2 The *Info cues* do not differ between subjects.
H3 For a given change request, the *modifiability* of MF and RD is related to the design approach.
H4 For a given design (MF/RD), the *modifiability* of a change is correlated to its perceived difficulty.
H5 Participant's performance on a change is correlated to the *modifiability* of that change.

H1 and H2 address face validity. H3 and H4 deal with concurrent validity. H5 assess predictive validity.

Relative to face validity (H1 and H2), space does not allow us to show all of the acts and ic identified. The acts varied among subjects, while the ic referenced in the acts were relatively consistent for a given change task. There is a subset of ic that are consistent across the subjects in the same group (MF or RD) for a given change. Therefore, the results support H2 but not H1.

We then measured the component complexity of the MF and RD designs for each change using all six measures shown in Table2 as below.

Table 2 Component Complexity

D	M1	M1'	M2	M2'	M3	M3'	CH
MF	21	21	15	15	14	14	**C1**
	89	64	58	47	52	42	**C2**
	498	114	263	84	235	79	**C3**
RD	21	21	15	15	14	14	**C1**
	45	45	50	50	43	43	**C2**
	90	70	76	64	62	52	**C3**

For modifiability the relative order of metrics is more meaningful than the absolute values. There are 9 comparisons for "relative modifiability", three across software systems, i.e. MF vs. RD for C1, C2 and C3 respectively, and six across changes, i.e. C1 vs. C2 vs. C3 for MF or RD. The results were consistent regarding relative modifiability for all 6 measures as follows (note that the lower component complexity, the higher modifiability.):

For each change:

C1: MF=RD; **C2**: MF<RD; **C3**: MF<RD

For each software system:

MF: C1 > C2 > C3; RD: C1 > C2 > C3

Our measurement results were confirmed by the difficulty level of the maintenance tasks determined a priori [10]. C1 is the easiest (highest modifiability) and C3 is the most difficult (lowest modifiability) for both MF and RD. When comparing across the designs, a detailed analysis (by experts) drew the same conclusion as our model: C1 had the same difficulty in both designs, but C2 and C3 were easier for RD than for MF. Therefore, H3 and H4 are supported.

We used two performance measures from the original study to test H5: total time (maintenance effort) and correctness score (the quality of the changed code). Effort and correctness describe the task performance from different but competing perspectives. We combined them into a single metric by dividing time by correctness. Spending more time or delivering lower quality increase the value of this new performance metric. Therefore, a higher score indicates poorer performance. Results showed that for all 10 subjects, the performance on CH1 is the best. This result is consistent with the relative modifiability results mentioned above. The performance on CH2 is better than the performance on CH3 for 8 of the 10 subjects. The two exceptions were both in the RD design group. Therefore, H5 is partially supported.

In summary, the results of case study supported the face validity of the info cues as model components and the concurrent validity of the model. The results also partially assured the predictive validity.

5. Conclusion

In this paper, we propose an analytical model that describes the characteristics of maintenance tasks and the analytical dimensions of modifiability. By applying task analysis theory, the proposed model captures the difficulty (complexity) inherent in the maintenance task independent of individual characteristics of maintainers. Thus, the model provides more validity and stronger generalization for software maintenance studies (e.g. effort prediction). The model also provides greater insight into the relationship between internal software attributes (e.g. structural measures) and external attributes of maintenance process (e.g. effort).

We will continue our research in the following directions:

Refining Model Components: we will look for common patterns in maintainers' behaviors to see if it is possible to identify more abstract acts that have construct validity. We will also extend the definition of info cues to include the control structure of the code in addition to PEs.

Exploring Measures for Component & Coordinate Complexity: currently we used traditional size measures to measure the information contained in the cues. In the future, we will look at other areas like information theory for information-related measures.

Controlled Experiments: we plan to conduct controlled experiments on a larger sample of subjects to further validate the proposed model.

Acknowledgements Work partially supported by NSF grant CCF-0438923

978-1-4244-4841-8/09/$25.00 ©2009 IEEE

References

[1] L. Erlikh, "Leveraging legacy system dollars for e-business", *IT Professional*, 2000. 2(3): pp. 17-23.

[2] ISO/IEC.2000. Information technology -Software product quality–Part1:Quality model. ISO/IEC FDIS 9126-1:2000(E)

[3] D.P. Darcy, C.F. Kemerer, and S.A. Slaughter, "The Structural Complexity of Software: An Experimental Test", *IEEE Transactions on Software Engineering*, 2005. 31(11).

[4] M. Polo, M. Piattini, and F. Ruiz, "Using code metrics to predict maintenance of legacy programs: A case study", in *2001 IEEE International Conference on Software Maintenance*. 2001. Florence, Italy: IEEE Computer Society.

[5] D.L. Lanning and T.M. Khoshgoftaar, "Modeling the relationships between source code complexity and maintenance difficulty", *Computer*, 1994. 27(9): pp. 35-40.

[6] L.C. Briand and J. Wuest, "Empirical Studies of Quality Models in Object-Oriented Systems", *Advances in Computers*, 2002.59: pp.97-166

[7] D. Kozlov et al., "Assessing maintainability changes over multiple software releases", *Journal of Software Maintenance and Evolution: Research and Practice*, 2008. 20(1): pp. 31-58.

[8] R.E. Wood, "Task Complexity: Definition of the Construct", *Organizational Behavior and Human Decision Process*, 1986. 37: pp. 60-82.

[9] E. Arisholm, D.I.K. Sjøberg, and M. Jørgensen, "Assessing the Changeability of two Object-Oriented Design Alternatives - a Controlled Experiment", *Empirical Software Engineering*, 2001. 6(3): pp. 231-277

[10] A. I. Wang and E. Arisholm, "The Effect of Task Order on the Maintainability of Object-Oriented Software", *Information and Software Technology*, 2009.51(2):pp.293-305.

978-1-4244-4841-8/09/$25.00 ©2009 IEEE