Assessing Software Product Maintainability Based on Class-Level Structural Measures

Hans Christian Benestad, Bente Anda, and Erik Arisholm

Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway
{benestad, bentea, erika}@simula.no
http://www.simula.no

Abstract. A number of structural measures have been suggested to support the assessment and prediction of software quality attributes. The aim of our study is to investigate how class-level measures of structural properties can be used to assess the maintainability of a software product as a whole. We survey, structure and discuss current practices on this topic, and apply alternative strategies on four functionally equivalent systems that were constructed as part of a multi-case study. In the absence of historical data needed to build statistically based prediction models, we apply elements of judgment in the assessment. We show how triangulation of alternative strategies as well as sensitivity analysis may increase the confidence in assessments that contain elements of judgment. This paper contributes to more systematic practices in the application of structural measures. Further research is needed to evaluate and improve the accuracy and precision of judgment-based strategies.

1 Introduction

Software engineering is a complex problem solving activity with conflicting quality goals [1]. Quality attributes that are difficult to measure may therefore receive little attention. A prime example of this is the quality attribute known as maintainability, defined as the capability of the software product to be modified [2]. Researchers have proposed a number of structural measures as indicators of maintainability. For example, the CBO (Coupling Between Objects) measure is defined as the count of classes to which a class is coupled. Two classes are coupled if one class uses methods or instance variables of the other. An increased CBO measure is hypothesized to indicate more difficult testing, maintenance and reuse [3]. Existing research on the application of structural measures have focused on analyzing historical data to build product specific prediction models that can identify error-prone classes or classes that will be difficult to maintain [4].

In some situations, the required focus of an assessment is on the software product as a whole, as opposed to on individual classes. For example, a software acquirer may want to assess the maintainability of a software system prior to acquisition, or a software provider may want to monitor the maintainability of a system during its construction. In this paper, we investigate the use of structural measures as indicators of quality attributes at the *system* level, focusing on maintainability in particular.

J. Münch and M. Vierimaa (Eds.): PROFES 2006, LNCS 4034, pp. 94 – 111, 2006.

Based on current practices reported in literature, alternative strategies for conducting system level maintainability assessment from structural measurements are identified and discussed. We explore possible strategies by applying them on four functionally equivalent software systems that were developed as part of a multi-case study conducted by our research group. In this study, we needed to rank the four systems with respect to likely future maintenance effort. The historical data required to build statistically based prediction models was not available; hence we had to use elements of judgment in our assessment. To increase the confidence in the assessment we cross examined results from applying alternative strategies, and from altering judgment based parameters. These techniques, called triangulation and sensitivity analysis are intuitive and straightforward; however our survey indicates that they are rarely put to use in practice. A possible explanation is that few guidelines exist for identifying and selecting alternative assessment strategies. The main contribution of this paper is to identify and structure possible assessment strategies, in order to support future measurement initiatives in their selection and combination of alternative strategies.

The remainder of this paper is structured as follows: Section 2 describes related work. Section 3 discusses alternative strategies for selecting and interpreting structural measures for system-level assessment of maintainability. Section 4 applies alternative strategies to our systems under study. Section 5 concludes.

2 Related Work

This section outlines related work, focusing on research that has used structural measures to assess software maintainability.

The *Goal/Question/Metric* paradigm [5] prescribes development of *measurement models* that links measurement goals to operational questions that can be answered by measuring aspects of products, processes or resources. The method is useful in order to ensure goal-driven and purposeful measurements, but must be combined with domain knowledge specific to the quality focus in question.

Hierarchical quality models, [2, 6, 7], relate external quality attributes (such as maintainability) to internal attributes and measures of internal attributes. These models provide a useful framework for designing measurement programs related to software quality, but still provide limited guidance in the specifics of selecting and interpreting structural measures.

Structural measures of software. Early structural measures included lines of code (LOC) and complexity measures by McCabe [8] and Halstead [9]. These measures are commonly adapted and used for object-oriented systems, with class-level interpretations such as “LOC per class”. The Maintainability Index (denoted MI) is a polynomial based on these measures that was suggested and validated as an indicator of maintainability by Oman [10]. With the advent of object-orientation, additional measures for size, complexity, inheritance, coupling and cohesion were suggested. The set of measures by Chidamber & Kemerer [3] (denoted CK) is among the most popular. The CK measures were hypothesized to be indicators of maintainability aspects, such as analyzability and testability, and have been empirically validated and used in a number of contexts [11].