Assessing Software Product Maintainability Based on Class-Level Structural Measures

Hans Christian Benestad, Bente Anda, 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. 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 [3, 11-17]. 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 [18, 19]. The Metrics for Object-Oriented Design (denoted MOOD) by Abreu [20] operate on the package and system level, and is like the CK measures intended to measure aspects of maintainability of object-oriented design.

Application of structural measures. A number of studies have applied structural measures to assess system-level design properties. The aim of the survey summarized in Table 1 was to characterize implicit or explicit assessment strategies that were employed in the studies, to be used as a basis for the discussions in Section 3. In order to facilitate the discussions, we subdivide an overall strategy into the categories of *selection, combination, aggregation and interpretation*. Alternative categorizations are possible, but we believe that these four concepts are fairly straightforward: *Selection* is the process of selecting the structural measures that best fit the purpose of the measurement initiative. For example, if the purpose is to investigate the decrease in overall size as a result of a re-engineering effort, the measure *lines of code* (LOC) may be selected. Most of the studies summarized in Table 1 used a small set of measures to conduct the assessment and did not explicitly describe how the measures were selected. The CK measures and the traditional LOC, McCabe and Halstead measures are typical selections. *Combination* is the process of providing a combined view of the values for the selected structural measures, for a given software artifact. The combined view may take many different forms, including a new numerical value, a vector of values, a qualitative statement or a visual chart. In the surveyed papers, most studies provide qualitative summaries or simple charting techniques, such as Kiviat diagrams. *Aggregation* is the creation of a derived measure at the system level, based on class-level measures. A simple example is to sum the size of all classes to produce a measure for system size. In the surveyed papers, aggregation of measures from the class level to the system level is typically based on summary statistics, while some studies visualize distributions to retain more information from the original dataset, and to pay specific attention to outliers. *Interpretation* is the assignment of a meaning to the measurement values with respect to the quality attributes of interest. An example of a simple interpretation strategy is to assume that smaller coupling values are advantageous with respect to future maintenance effort. Most interpretations in the surveyed papers are based on assuming such a preferred direction of measurement values, then comparing values from different versions of the same product, or comparing values from products with similar properties. Discussions of possible strategies and justifications of choices are provided to a very limited extent in the surveyed papers.

Table 1. Summary of survey on current practices on system level assessment based on structural measures. Where object-oriented languages are not used, we consider aggregation from the file-level and function-level

Study	Purpose of study	Selection of metrics	Aggregation	Combination	Interpretation
Sharble& Cohen [21], 1993	Compares two alternative design methodologies, by developing two functionally equivalent products	CK+three additional measures	Sum, outlier detection, plots of class vs. value	Inspection of related measures, Kiviat charts	Compare values between two systems
Chidamber & Kemerer [3], 1994	Illustrates the C&K measures by comparing two unrelated object-oriented systems	CK	Summary statistics, outliers, histograms	Suggests external attributes related to measures	Compare values between two systems
Harrison *et al.* [22], 1996	Compares the object-oriented to the functional paradigm by collecting process and structural measures on functionally equivalent software	LOC, functions called and defined, function depth	Statistical distribution	Inspection of related measures	Compare distributions using statistical tests
Drake [23], 1996	Describes the quality assessment of software systems at National Security Agency (NSA), and the use of structural measures to assess the effect of re-engineering initiatives	Halstead, McCabe, LOC	Per function analysis, identification of hot spots	Inspection of key measures, Kiviat charts	Threshold values computed from sample of existing source code.
Mayrand [24], 1996	Describes how 20+ software products have been assessed as part of a software acquisition process	Halstead, McCabe, LOC inspired	Identification of hot spots	Not reported	Threshold values based on local experience
Harrison *et al.* [25], 1998	Theoretically validates the MOOD measures, and illustrates design assessment of 11 software systems	MOOD	MOOD-factors operate at system level	Not reported	Judgment of values based on recommendations
Barnard [26], 1998	Suggests a reusability index based on structural measures and assumptions of actual reusability	CK	Plots of class vs. value	Suggests an expression indicating reusability	Threshold values based on heuristics
Schroeder [27], 1999	Describes how to use structural measures in the deployment of measurement programs	McCabe, LOC, basic OO measures	Count number of abnormal measures	Count number of abnormal measures	Threshold values from 90% percentile of existing systems
Briand& Wüst [28], 2001	Describes a method for comparing functional equivalent systems using a combination of static code analysis and analysis of probability of change	Extensive set of OO class-level measures	Weighted average (by likelihood of change)	Inspection of related measures	Compare distributions using statistical tests
Saboe [29], 2001	Determines the readiness for release	Halstead, McCabe, LOC	Mean values	MI,Kiviat charts	Threshold values, predefined by MI
Stamelos *et al.* [30], 2002	Investigates the quality of open source products	Halstead, McCabe, LOC inspired	Count number of modules at different quality level, summary statistics	Weighted sum to produce quality indicators	Threshold values defined by tool used to categorize modules
Bansiya and Davis [6], 2002	Establishes a hierarchical quality model based on structural measures	QMOOD [6]	Total count of classes and hierarchies, mean values	Weighted sum to produce quality indicators	Trend in quality indicators between versions
Ferenc *et al.* [31], 2004	Studies the evolution of a large, open source system	CK	Frequencies, summary statistics	Inspection of related measures	Trend in rate of classes having specific values between versions

3 Strategies for System Level Assessment Based on Structural Measures

Table 2 summarizes options that were identified for each sub-strategy. The options are not mutually exclusive. Furthermore, sub-strategies can in principle be applied in any order. In particular, an important decision to be made is the order in which aggregation and combination are performed: Measures may be combined at the class level before this combined class-level measure is aggregated to the system level. Alternatively, each class-level measure may be aggregated to the system level before the system-level measures are combined.

Table 2. Sub-strategies and options for creating a system level assessment strategy

Selection	Aggregation	Combination	Interpretation
1. Tool-driven selection 2. Pre-defined set of measures 3. Evidence-based 4. Statistical analysis of structural measures 5. Univariate regression analysis	1. Summary statistics 2. Distribution analysis 3. Outlier analysis 4. Visualization: Histograms, box plots, pie charts, scatter plots	1. Inspection of measures related to common quality attribute. 2. Multi criteria decision aid techniques (e.g. weighted sum and profile comparison) 3. Multivariate regression analysis 4. Visualization: Kiviat charts, Chernoff faces	1. Relative 2. Thresholds 3. Trend analysis 4. Prediction models 5. Visualization: Pie charts, line charts

The concepts of aggregation, combination and interpretation were explained in Section 2. More formally, if *n* measures are collected for each of *k* classes, we have:

m_{ij}	Measure value for measure *i*, class *j*.
$C_j=f_c(m_{1j}..m_{nj})$	Combined view[1] of a class *j*.
$A_i=f_a(m_{i1}..fm_{ik})$	Aggregated view for a measure *i* to a higher level of granularity
$M_1=f_{m1}(C_1..C_k)$	Aggregation of combined views to the system level
$M_2=f_{m2}(A_1..A_n)$	Combination of aggregated views at the system level
$I=f_I(m_{ij} \vee C_j \vee A_i \vee M_1 \vee M_2)$	Interpretation of one or more measures or views

The following sections discuss the options summarized in Table 2 in further detail.

3.1 Selection

Tool-driven selection is probably a common practice when structural measures are selected. For example, we believe that part of the popularity of the LOC measure can be contributed to the wide availability of tools that can count lines in text files. More advanced tools, such as Borland Together [32], collect a wide selection of structural measures. The selected measures should support the goals and questions of the specific measurement initiative, regardless of the sophistication of the tools employed.

A number of *pre-defined sets of measures* have been suggested by researchers. These sets are hypothesized to be indicators of aspects of maintainability, and they have to a varying degree been empirically validated. Popular sets include the CK measures, the MOOD measures and the MI, all briefly described in Section 2. Most of the studies summarized in Table 1 use a pre-defined set of measures.

With *evidence-based selection* practitioners search for empirical evidence on questions that are similar to their own, using the principles of evidence-based software engineering [33]. For example, the survey by Briand & Wüst [4] provides concrete recommendations on types of measures to consider as indicators of different quality attributes. Although the infrastructure and extent of empirical knowledge is not yet sufficient for wide adaptation of the evidence-based selection strategy [34], mature organizations may still gain from an evidence-based strategy because it better supports changing goals and questions, than do the use of a pre-defined set of measures.

[1] The term *view* is used because the operations do not necessarily produce a single output value, but can also take the form of vectors of values, qualitative statements or visual charts.

Selection based on *statistical analysis of structural measures*, addresses the weakness of the former strategies where the selections are not based on the particularities of the systems under study. Such analysis can address the *discriminating power* and *orthogonality* of structural measures: If the interpretation of measure values is based on comparison with other versions or systems, we consider it advantageous to select measures that can discriminate between the designs in question. For example, if coupling by data abstraction (attribute has class as type) is a rare construct in the analyzed systems, this measure would be a non-optimal choice for measuring coupling. Simple summary statistics can be used to identify measures with low variance or with few non-zero observations. Interpretation is simplified if *orthogonal* measures are selected, meaning that the same design aspects are not measured more than once. *Principal component analysis* (PCA) [35] is a statistical method that addresses both the discriminating power and orthogonality.

If it is possible to collect historical data that are direct or indirect measures of the quality attribute of interest, a *univariate regression analysis* can be conducted. The purpose of the analysis is to obtain historically based evidence on a relationship between a specific structural measure and a given quality attribute. This strategy is used in studies surveyed by Briand and Wüst [4].

In practice, a specific strategy for selecting structural measures may contain elements of several of the above described options. Availability of tools may impose limitations on which structural measures can be considered. The availability of tools may be limited e.g. due to lack of support for a given technological platform. Furthermore, a combined strategy is to use statistical analysis of structural measures to adjust a predefined set of measures to better be able to discriminate between systems. An example of a combined strategy is provided in Section 4.1.

3.2 Combination

Structural measures should ideally measure one and only one aspect of design. The complex concept of maintainability can be expected to be influenced by several aspects of design. For example, the effort required to comprehend a class may be influenced by size, complexity and coupling. A combined view must be created for these measures to support the assessment of the required comprehension effort. It is inherently difficult to merge largely unrelated measures into a derived measure that can serve as an indicator of a quality attribute of interest. However, some formal or informal combination is required if several structural measures influence on the quality attribute.

Some of the studies summarized in Table 1 perform combination by inspecting measures that are believed to influence a common quality attribute, for systems or versions of a system that are to be compared, c.f., [21, 31]. If there is a consistent pattern of more desirable measure values for System A than for System B, the interpretation is that System A is more desirable than System B with respect to the chosen quality attribute (see Section 3.4 regarding interpretation).

Using the *weighted sum* of several structural measures is an alternative strategy. A weight is assigned to each measure to account for differences in measurement units and importance. Then the sum of the products of the weight and each measure value is computed. The complicating aspect of different measurement units are frequently resolved by converting the measure values to a common, ordinal scale with scores from e.g. 1 to 6. There are two main problems with this strategy: First, from a measurement theoretical point of view it can be questioned whether weighing and summing are legal operations. Second, the derived measure is difficult to interpret as a standalone measure. Third, the weighted sum method is *compensating*, meaning that negative scores can be completely compensated by positive scores, making relative comparison between software artifacts difficult. Bansiya and Davis [6] use a weighted sum strategy as part of establishing a hierarchical quality model that links internal design properties to external quality attributes. Stamelos *et al.* [30] reports from the use of the tool Tau Logiscope [36], which uses a similar strategy to produce combined measures meant as indicators of maintainability aspects. The Maintainability Index [10] combines three (or four, in one version) traditional code measures into a polynomial to produce a single valued indicator of maintainability.

The weighted sum strategy is well known, and even used in every day situations. The technique belongs to the larger class of multi criteria decision aid (MCDA) techniques. Morisio *et al.* [37] have proposed the use of another MCDA technique, known as *Profile comparison*, in the context of software artifact evaluation. Profile vectors are constructed from threshold values for a set of structural measures. For example, if *Very High* values for measures m_1, m_2 and m_3 are judged to start at 20, 30 and 100, respectively, the profile vector is:

```
Very High =[m1>=20, m2>=30, m3>=100]
```

The performance vector, i.e., the actual measures for a software artifact is iteratively compared to each profile vector, testing whether a “sufficient” majority supports the classification, and classification is not “strongly” opposed by a minority. For example, the performance vector

```
ClassA =[m1=22, m2=20, m3=150]
```

is classified as Very High using the simple rule that the majority of the measures should support the classification. Weights can be assigned to reflect relative importance between the measures, and specific veto-rules can be specified. The use of this technique is illustrated in Section 4.3. Other MCDA techniques, such as the Analytic Hierarchy Process (AHP), have been suggested in the context of COTS evaluation [38], and can be a candidate for further exploration in the context of assessments based on structural measures.

With *multivariate regression analysis,* historical data can be used to construct models that predict, e.g., error-prone classes or classes that will be difficult to maintain. A common strategy is to first use Principal Component Analysis (PCA) and univariate regression to identify candidate measures to be included in the model. Employing various variable selection heuristics, a multivariate prediction model is built for predicting the quality attribute of interest. The technique is described and used in [39].

Visualization. *Kiviat charts* are frequently used to support combinations that aim at comparing systems or comparing against threshold values. Drake used the shape of Kiviat charts to identify the “Albatross syndrome”, which was a recurring pattern of undesirable values [23]. An example of the use of Kiviat charts is also provided in Fig. 1 in Section 4.2. With simple goals, such as “Check that all aggregated values are below a threshold” or “Does Product A have consistent lower scores than Product B” this technique can be effective. *Chernoff faces* is an alternative technique that can be used for visualizing multi-dimensional data: Each dimension is represented by a facial feature, in order to take advantage of the capability of the human brain to recognize human faces.

If it is possible to collect historical data that are acceptable measures of the maintainability of the system, the regression-based methods are likely to produce the most trustworthy results. This is the strategy of most of the studies in the quality model survey by Briand&Wüst [4]. Since these models usually operate on the class level, combined measures must still be aggregated according to some of the techniques discussed in Section 3.3 to produce a system level measure. However, as already pointed out, historical data may not be available if the goal is to assess a system for aspects of maintainability while the system is still under development. Techniques that include elements of judgment must be used in these situations.

3.3 Aggregation

Summary statistics are simple and standard techniques to describe a larger data set using one or a few numbers, see for example [3, 21, 30, 31]. However, there are many choices to be made when calculating the summary statistics, as discussed below.

Sum values vs. central tendency. In the study by Sharble and Cohen [21], the sum of the measure values was used as a main aggregation method to compare the maintainability of two systems developed using two alternative design methods. This approach may be inappropriate. The relationship between the structural measures and maintainability may be nonlinear, but using the sum assumes a linear relationship. Furthermore, by using the sum, the measures are confounded with size, because larger systems will have higher measure values simply due to size. Obviously, size (e.g. the number of classes) is not unimportant, and can be a candidate as a separate measure, c.f., [6]. We thus believe that measures of central tendency (i.e., the mean and median) are more appropriate aggregation operators than *sum,* at least in the context of maintainability assessments.

Mean values vs. median values. Mean value tends to be the most frequently used measure of central tendency in the studies summarized in Table 1. In some cases, the median value would have been a preferred choice: Distributions of structural measures tend to be skewed to the left; hence the median is typically lower than the mean. When complementing the analysis with a specific analysis of the rightmost part of the distribution (high values and outliers) very high values will not be accounted for twice if the median is used instead of the mean. However, mean value remains the preferred choice when aggregation into one single number is necessary, because the important information resident in very high measures is hidden by median values.

*Dispersion and distribution analysis***.** With increasing variance of complexity measures, a greater part of the classes have higher or lower complexity. Using the same reasoning as above, the penalty of analyzing a very complex class may more than outweigh the gain of analyzing the very simple class. Measures of dispersion are therefore important to consider. Low variance indicates a balanced design, with functionality and complexity distributed along a broad set of classes. One option is to calculate the variance or standard deviation of each measure. However, more information is retained from the original data set by creating frequency tables: A set of intervals are defined and the number of classes within each interval is counted. This is the predominant method for analyzing the nature of the distribution, with specific attention to high values, see e.g. [3, 31].

Outlier analysis: Outliers in measure values can be identified by using scatter plots, or with statistical methods. They may contain important information about the nature of design. There may be good reasons for a few classes in a system to have very high measures for individual structural attributes. For example, the use of some widely accepted design patterns can result in high measures of attributes, methods, or coupling. To be able to conclude whether the existence of an outlier can be defended, one may resort to the costly procedure of manual code inspection. Classes that are outliers with respect to several independent measures can more immediately be assumed to be an undesirable aspect of system design, even without manual inspection. Principal component analysis, discussed in Section 3.1, can be helpful in identifying the dimensions that should be included in such analysis. An example of detection of two-dimensional outliers using scatter plots is provided in Fig. 2.

Visualization: *Histograms* are frequently used for illustrating tabulated frequency of values of structural measures. *Box plots* provide a more compact technique to visually compare dispersions of systems. An example of the use of box plots is provided in Fig. 3.

In summary, using the sum or the mean values of structural measures at the class level provide very approximate measures of the overall design of a system. Unless linear relationships exist between the measures and the external quality attributes of interest, the sum or mean values may even be misleading. The median combined with measures of the variance and outliers might thus be a better choice in many situations. By using frequency analysis, more information from the original dataset is retained, and it is possible to take different kinds of non-linear relationships into account. However, retaining more information in the aggregation step will increase the complexity of the interpretation, discussed in the next section.

3.4 Interpretation

Interpretation is the assignment of a meaning to the measurement values with respect to the quality attribute of interest. Interpretation can occur as a final step after the combination and aggregation of measurement values, in which case the aim is to answers the questions related to some quality attribute. It can also occur at the class level, for example if the profile comparison technique discussed in Section 3.2 is employed. The relationships between the internal measures and the external quality attributes can only be established to a certain extent. The tolerated level of uncertainty may have an impact on how the measure values can be interpreted. For example, the uncertainty should be low if the goal is to test some contractually specified requirement to maintainability, hence assessment strategies that include elements of judgment may not be appropriate in this situation. However, if the goal is to support a development team in making reengineering decisions, judgment based strategies may provide considerable value.

Relative interpretation is the most basic principle for establishing relationships between internal and external characteristics of a software artifact. By assuming, say, that lower values are more desirable than high values with respect to maintainability, it is possible to rank software artifacts regarding maintainability. The assumption of a purely increasing or decreasing function may not hold in all cases: A system with deep inheritance trees may be difficult to maintain, while a system with no use of inheritance may not exploit useful features of object-oriented languages. Also, with a purely relative interpretation, the significance of observed differences cannot be determined. Finally, if individual measures show inconsistent patterns for a given software artifact, it can be difficult to conclude.

With *thresholds values*, the range of possible measurement values is sub-divided into intervals that are given specific interpretations. For example, intervals in a frequency table can be assigned names such as Good, Acceptable, Suspicious and Unacceptable. This type of interpretation is used in tools, such as Tau Logiscope [36]. The overall assessment can pay specific attention to the relative and absolute number of

Suspicious and Unacceptable classes. The main problem with the method is to decide on interval thresholds. A similar strategy can be part of the profile comparison technique, described in Section 3.2. Outlier detection can be considered a special case, in which attention to extreme values are paid. Some tools, such as Borland Together [32], support the detection of outliers by highlighting classes or packages that exceed a predefined or user configured threshold value.

If measures from earlier versions of the product are available, *trend analyses* can provide useful insights. A system can be expected to be more resilient to change if measure values are stable between releases.

If *prediction models* based on historical data are built, the maintainability can be more precisely quantified. For example, based on historical data it may be predicted that the probability of a fault in a class doubles, if a measure of complexity increases by 50 %. Attempts have been made, most notably with the Maintainability Index [10], to create a model that is more generally applicable than local prediction models. It is difficult to argue in favor of the general validity of such a formula. However, adapted and used in local context, considerable value has been reported from using it [29, 40].

Visualization: Pie charts are frequently used to illustrate the relative number of classes receiving a specific classification when using threshold values. Line charts can be used to illustrate trends in measurement values. An example of the use of pie charts is provided in Fig. 4.

All studies summarized in Table 1 use relative interpretation as an underlying principle, while a majority of the studies use threshold values as part of their interpretation [23, 24, 26, 27, 29-31]

3.5 Confidence Assessment

The discussion above shows that there are many uncertainties related to an assessment that is based on structural measures. This is not surprising, since the attempt is to draw conclusions on complex external quality attributes. General methods exist for assessing the confidence in methods or models that include sources of uncertainty.

The idea behind *triangulation* is that one can be more confident with a result if different methods lead to the same result. The options described in Table 2 can support the creation of alternative assessment strategies, the results of which can be cross examined to increase the confidence in the conclusions. With inconsistent results, it may not be possible to draw firm conclusions. In Section 4.2 and Section 4.3, an assessment is conducted with an aggregation-first and combination-first strategy, respectively.

Sensitivity analysis is a procedure to determine the sensitivity of the outcomes of a model to changes in its parameters. If a small change in a parameter results in relatively large changes in the outcomes, the outcomes are said to be sensitive to that parameter. The judgment based strategies that have been discussed in Section 3 can be regarded as models in where parameter settings are subject to uncertainty. For example, threshold values that are set by judgment during interpretation of measurement values can be varied within a range of reasonable values. If conclusions are insensitive to the threshold values used, more confidence can be put in the results. The number of alternative assessments will grow quickly when combinations of parameters with wide ranges of reasonable parameter values must be tested. However, this cost can be significantly reduced by automating the execution of the models.

The studies summarized in Table 1 do not report the use of triangulation or sensitivity analysis. If model evaluation does not occur, it is difficult to put trust in the results. We provide examples of triangulation and sensitivity analysis in the next section.

4 Assessment of the DES Systems

The overall goal of the Database of Empirical Studies (DES) multi-case study was to investigate differences in development style of software providers, and their effects on software projects and products. In the study, we contracted four Norwegian software houses to independently construct a software system, based on the same requirement specification. Based on incoming proposals and company descriptions, we selected contractors that were likely to represent four dissimilar development cultures. Agreements and interaction between the client (us) and the contractors adhered to regular commercial and professional standards. The systems manage information about empirical studies conducted by our group, and emphasize is on storage and retrieval of such information. Each of the contractors spent between 431 and

943 man-hours on the project. We received consistent feedback that the projects closely resembled the development of an increment of a larger software product using an iterative development style.

Our strategy for assessing and comparing the DES systems is based on the discussions in the previous section. The goal for the assessment is to rank the four systems with respect to future maintenance effort, and to explore the strategies described in Section 3. Since we do not yet possess rich empirical data from the maintenance phase, options involving regression analysis are not used; instead parts of the assessment are qualitative and judgment-based. We apply triangulation and simple sensitivity analysis as described in Section 3.5 to increase confidence in the results.

4.1 Selection of Measures

Class-level measures that are hypothesized and partly validated to influence maintainability are referenced in Section 2. For tool support, we surveyed commercial tools, open source tools and tools from academia, and chose two tools that in sum covered a total of 71 relevant measures (M-System from Fraunhofer IESE and JHawk from VirtualMachinery). Ahead of this decision, we observed that different tools did not necessarily give consistent results for the same measures, which is likely to be due to differences in the detailed interpretation of the definition of the measures. We therefore chose to limit the number of different tools used, and ensured that related measures (i.e., measures that depend on common underlying information) were collected by the same tool.

Based on the discussions in Section 3.1 we investigated four alternative strategies for reducing the initial set of measures collected by the tools. An appropriate set of measures for our goal would be a minimal set that measures the dimensions of design that influence maintainability.

1. *Predefined set of measures*. The CK set is probably the most popular predefined set of measures. One problem with this set is that it includes only one measure for the concept of coupling, while it is known that fan-out (import) coupling has different effects than fan-in (export) coupling. Also, the LCOM measure has been shown to confound with size, and is not necessarily an appropriate measure of the concept of cohesion. The measures included in the CK set are shown in column "CK" in Table 3.
2. *Evidence-based*. We adjust the CK set of metrics to overcome the problems that were indicated above. We substitute the CBO coupling measures with separate measures for export coupling and import coupling (PIM, PIM_EC). LCOM is substituted by TCC, which is a normalized cohesion measure that has more discriminating power and is less influenced by size [41]. Also, we add the LOC measure as a straightforward measure of size, which is not included in the CK metrics. The selected measures with this strategy are shown in column "EV" in Table 3.
3. *Principal component analysis*. With the two above strategies it is uncertain whether the selected measures are orthogonal and have the discriminating power desired for the systems under study. We performed a principal component analysis to support the identification of such measures, and select the measure with highest loading for each of the first eight principal components. We experienced that it was difficult to interpret some of the resulting principal components as distinct dimensions of design. Also, we found the analysis to be sensitive to the selection of input measures. The selected measures with this strategy are shown in column "PCA" in Table 3.
4. *Combination of the above*. In the last approach we fine-tune the selection from the CK and evidence-based strategy by making sure it does not contradict the PCA analysis. The measures TCC, DIT, NOC and WMC1 had high loadings on components that could be interpreted as "normalized cohesion", "inheritance height", "inheritance width" and "size and complexity", respectively. We therefore retain these as selected measures. For coupling measures we replace PIM and PIM_EC with OMMIC and OMMEC because the PCA indicated that the latter import/export coupling pair measures more distinct aspects of design. The LOC and WMC2 measures are removed, since they load moderately on the first principal components, already represented by the WMC1 measure. Instead, we choose to give double weight to the WMC1 measure in the following analysis. Since inheritance is not a widely used mechanism in the systems under study, we choose to halve the weight on each of the two inheritance measures. The selected measures, which constitute our final selection, are shown in column "Final" in Table 3.

The collected measures for the 71 candidate measures and the detailed results from the PCA are available at: http://www.simula.no/departments/engineering/projects/ooad/StudyData/StudyDataProfes2006/.

Table 3. Analysis of DES systems: Selection of measures using four alternative strategies

Measure	Source	Short description	CK	EV	PCA	Final
LOC	Trad.	Lines of code		X		
WMC1	[3]	Number of methods in class	X	X	X	X
WMC2	[3]	Cyclomatic complexity. Number of possible paths	X	X		
CBO	[3]	Coupling between objects	X			
OMMIC	[17]	Call to methods in unrelated class			X	X
OMMEC	[17]	Call from methods in unrelated class				X
IH_ICP	[14]	Information-flow based coupling			X	
DAC_	[13]	Data abstraction coupling			X	
PIM	[42]	Polymorphically invoked methods		X		
PIM_EC	[42]	Polymorphically invoked methods, export version		X		
OCAEC	[17]	Class used as attribute type in other class			X	
NOC	[3]	Number of children	X	X		X
NOD	[15]	Number of descendants			X	
DIT	[3]	Depth of inheritance tree	X	X		X
NOA	[11]	Number of ancestors			X	
TCC	[41]	Tight class cohesion		X	X	X
LCOM	[3]	Lack of cohesion	X			

4.2 Aggregation First Strategy

We here present the results of a quantitative and qualitative assessment, using an aggregation-first strategy. Summary statistics are provided in Table 4. Kiviat charts of mean values are provided in Fig. 1.

Table 4. Analysis of DES systems: Summary statistics. For Mean and Standard deviation, values that deviate by more than 30% from Total values are in italics

	Mean						Median					
	WMC1	OM MIC	OM MEC	DIT	NOC	TCC	WMC1	OM MIC	OM MEC	DIT	NOC	TCC
System A:	6.9	*7.7*	*7.7*	*0.46*	0.46	*0.26*	3.0	2	0	0	0	0.05
System B:	7.8	5.3	5.3	0.75	0.59	0.17	*6.0*	0	0	1	0	0
System C:	*11.4*	*8.6*	*8.6*	*0.0*	*0.0*	0.20	*8.0*	0.5	3	0	0	0.12
System D:	*4.9*	4.7	4.7	*0.83*	*0.76*	*0.11*	4.0	2	1	0	1	0.0
Total:	7.1	5.8	5.8	0.67	0.57	0.17	4.0	1.0	0.0	1.0	0.0	0

	Standard deviation						Sum					
	WMC1	OM MIC	OM MEC	DIT	NOC	TCC	WMC1	OM MIC	OM MEC	DIT	NOC	TCC
System A:	11.2	15.8	*20.6*	0.50	2.75	0.37	435	486	486	29	29	n.a.
System B:	10.3	11.8	15.6	0.81	2.37	0.31	1265	852	852	120	95	n.a.
System C:	*12.5*	*25.0*	*16.0*	0	0	0.23	273	206	206	0	0	n.a.
System D:	*4.5*	14.1	*10.1*	0.54	*3.81*	0.22	473	451	451	80	73	n.a.
Total:	9.6	14.4	15.6	0.69	2.83	0.30	2446	1995	1995	229	197	n.a.

Mean values: For mean values, the most distinct pattern can be observed for System C, which has relatively high values for size and complexity (WMC1) and coupling (OMMIC, OMMEC). Zero-value for the inheritance measures indicates that inheritance is not used in this system. Despite of a cohesion value (TCC) around average, this analysis is a first indication that the design of System C is non-optimal. System D has a low measure for WMC1 (considered desirable), but relatively high measures of inheritance, and low cohesion. System A has large coupling values (considered undesirable) and relatively low inheritance depth and high cohesion (considered desirable), while System B has no conspicuous mean values. This analysis indicates that System D will be the most maintainable system, while System C will be the least maintainable. It is difficult to rank System A and B, but since coupling measures are more than 30% larger than average for System A, we would rank A ahead of B.

Median values: We observe that due to the relatively large number of 0-values for all measures but WMC1, the median values become difficult to interpret. For WMC1, the pattern observed from the mean values recurs.

Standard deviation: System C has relatively large standard deviations for WMC1, OMMIC and OMMEC. System D has a small standard deviation for WMC1. This indicates that size and complexity is well distributed across the classes in System D, but not for classes in System C. System A has a rather large standard deviation for export coupling (OMMEC). There are no conspicuous values of standard deviation for System B. The suggested ranking from above is thus further supported from this analysis, however it is still difficult to distinguish between System A and System B.

Sum values: The sum values largely reflect the total size of the systems, measured in number of classes. The systems A, B, C and D contain 63, 162, 24 and 96 java classes, respectively. The low number of classes of System C introduces an uncertainty regarding the firm conclusion about this system from above. System B contains more than 2.5 times the classes than do System A. We interpret this as significant, given the close evaluation of these systems above. System B contains slightly more desirable code, but we rank it behind System A because 2.5 times the amount of classes must be comprehended and maintained.

Outliers: The box plots in Fig. 3 show that System D and System C contain extreme outliers for the OMMIC measure. The names of these classes, *StudyDAO* and *DB* respectively, indicate that the first uses the data access object pattern, while the latter is likely to be a convenience based grouping of database access. System B contains one extreme outlier for the OMMEC measure, while System A contains one extreme outlier for the WMC1 measure. The name of these classes, *StudyForm* and *ObjectStatementImpl* respectively, indicates that these values are acceptable: A class handling a complex GUI may have many conditional paths, while a class supporting object persistence may be heavily used by other classes. The two-dimensional scatter plots in Fig. 2 show that System B, C and D contain one class that can be considered an outlier with respect to both size and complexity (WMC1) and import coupling (OMMIC)[2]. It counts to the advantage of System A that the system contains no such two-dimensional outliers.

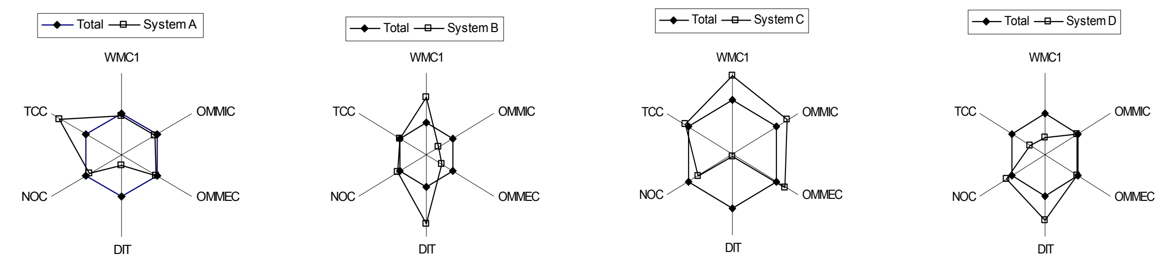

Fig. 1. Analysis of DES systems: Kiviat charts, comparing mean values of each system to those of the concatenation of the systems

[2] These two measures were used in the analysis of outliers since they are representatives of the first two principal components from our PCA, which explains a large part of the variability of the total data set, c.f., 35. I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer-Verlag, 2002..

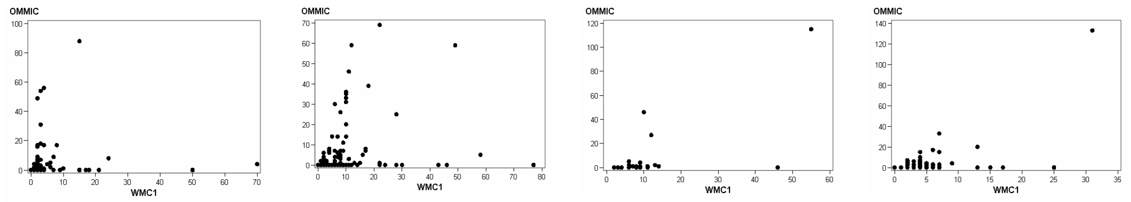

Fig. 2. Analysis of DES systems: Scatter plots of WMC1 versus OMMIC in the order System A, System B, System C, System D

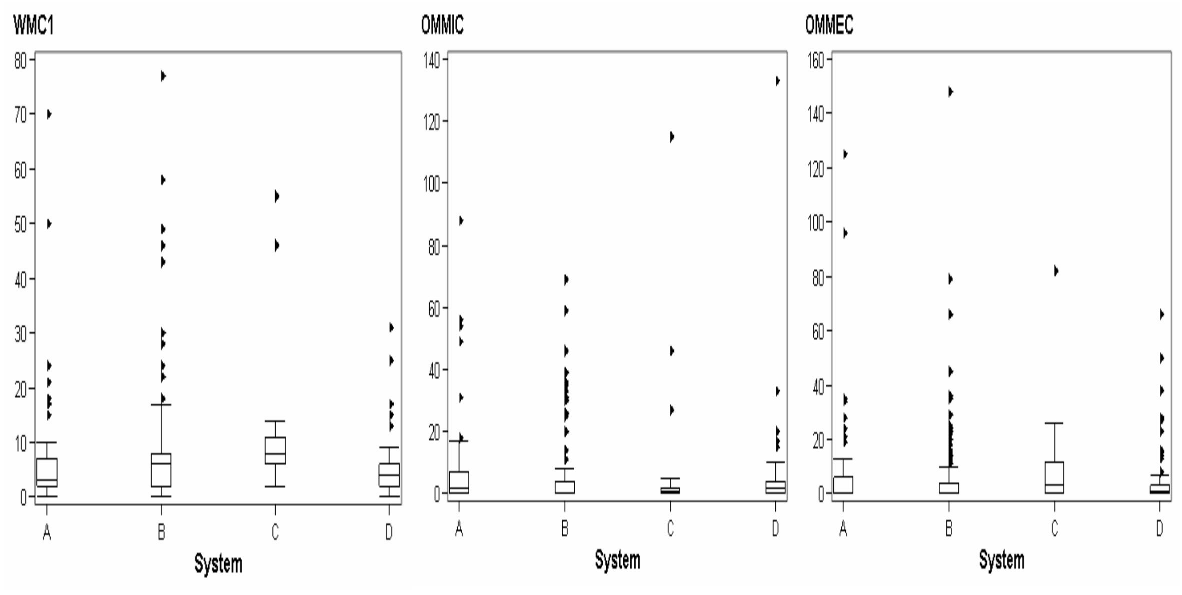

Fig. 3. Analysis of DES systems: Box plots for three structural measures

4.3 Combination First Strategy

We conduct this analysis by combining measures at the class level, and then aggregating the combined values using a frequency table. Finally, the frequency table is interpreted by putting most weight on high measures.

Class-level combination: We use a simple version of the profile comparison method described in Section 3.2, and create four *profile vectors*, labeled Low, Average, High and Very High, see Table 5. The interval limits are calculated from the 0 to 50 percentile, 50 to 75 percentile, 75 to 95 percentile and above 95-percentile of the concatenation of all classes. We then construct the 345 performance vectors from the 345 classes in the systems, of which an excerpt is provided in Table 6.

Table 5. Analysis of DES systems: Profile vectors. Weight of measure in parentheses

	WMC1 (2)	OMMIC (1)	OMMEC (1)	DIT (0.5)	NOC (0.5)	TCC (1)
Low	0-4	0	0	0-1	0	0.33+ or 0
Average	5-8	1-4	1-4	n.a.	n.a.	0.14-0.33
High	9-22	5-27	5-27	2	1-2	0.08-0.14
Very High	>23	>28	>28	>3	>3	<0.13

Table 6. Analysis of DES systems: Performance vectors for two example java-classes

	WMC1	OMMIC	OMMEC	DIT	NOC	TCC	Aggregated
Study	43 (VH)	0 (L)	79 (VH)	1 (L)	0 (L)	0.07 (VH)	Very High
Search	3 (L)	0 (L)	2 (A)	0 (L)	1 (H)	0 (L)	Low

The 345 performance vectors are compared against the profile vectors. The comparison criterion used is "The weighted sum of the criteria supporting the classification should be larger than the weighted sum opposing the classification". Class "Study" in Table 6 has three Very High values and three Low values. The weighted sum of the Low values is 2. The class must therefore be classified as Very High to meet the comparison criterion. Class "Search" has four Low values with a weighted sum of 4.5, and can therefore be classified as Low.

To be able to interpret the results showed in Table 7, we assume that the classifications can be read as Good, Acceptable, Suspicious and Undesirable. 91.7% of the classes of System D are Good or Acceptable, while only 2% are Undesirable. For System C, one third of the classes are Suspicious or Undesirable. These observations support the conclusions for these systems from Section 4.2. System A and B fall between these extremes, with System A having slightly more desirable classifications than System B. The absolute numbers make the ranking clearer: 35 classes in System B are classified as Suspicious or Undesirable, while 10 classes in System A receive this classification. This analysis indicates that the ranking of the systems with respect to likely future maintenance effort is: D, A, B, C (least effort mentioned first).

Table 7. Analysis of DES systems: Categorization of combined class level measures

	System A	System B	System C	System D
Low	65.1% (41)	53.7% (87)	29.2% (7)	60.4% (58)
Acceptable	19.0% (12)	24.7% (40)	37.5% (9)	31.3% (30)
High	12.7% (8)	18.5% (30)	25.0% (6)	6.3% (6)
Very High	3.2% (2)	3.1% (5)	8.3% (2)	2.1% (2)

Sensitivity analysis. The classification procedure was automated using Microsoft Excel and a Visual Basix macro. We could therefore easily re-conduct the analysis to investigate the sensitivity for threshold values, weights and classification criteria. Classification was expected to be sensitive to these variations, but we obtained consistent results as far as ranking between the systems was concerned. The analysis was most sensitive to the weighing factors: With equal weight for all measures, it was difficult to differentiate between the systems A, B and D. However, we consider it fair to put less weight on the two inheritance measures, since inheritance is not a widely used mechanism in the systems under study.

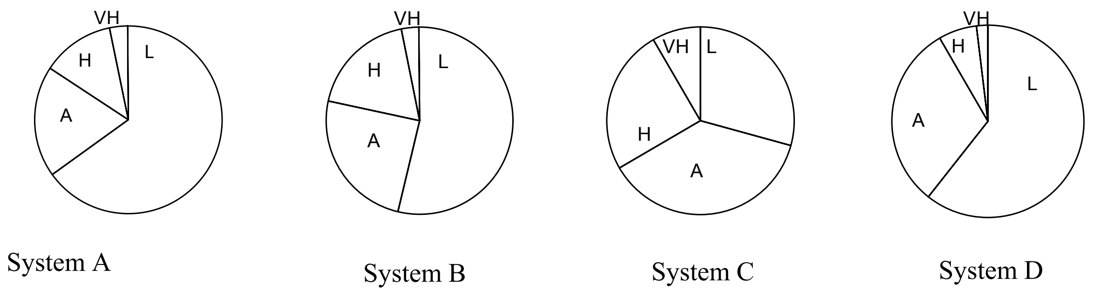

Fig. 4. Analysis of DES systems: Pie charts. Distributions of Low (L), Acceptable (A), High (H) and Very High (VH) classes

4.4 Summary of DES Analysis

Two parallel strategies were used for the DES analysis. The first strategy combined measures at the system level, while the second strategy created a derived measure at the class level. The latter strategy may be more intuitive for a developer who perceives the *class* as the main unit of analysis, change and testing during maintenance.

Systems C and D were consistently ranked lowest and highest, respectively. However, the significance of the small size of System C, measured in number of classes, leaves us with an uncertainty: It is quite possible that for some maintenance tasks, the effort involved in changing System C will not exceed that of the other systems. With the first strategy it was difficult to judge between System A and B, but we ended up with ranking A before B due to smaller overall size, and the absence of multi-dimensional outliers in System A. This ranking between System A and B was supported by the second strategy, which indicated a less desirable classification of classes in System B than in System A. The difference between the two systems was more evident when absolute number of classes was considered in place of relative number of classes for a given classification.

The accuracy of the described predictions of maintainability will not become evident until empirical data is collected from the maintenance phase. However, we asked an experienced consultant, who had not been involved in neither the development projects nor the research, to assess the code with respect to maintainability using his own experience from maintaining object oriented code. The results were consistent regarding System C and D. The expert ranked A before B, largely because of the difference in size.

5 Conclusion and Further Work

We have investigated strategies for collecting structural measures at the class level to perform system level assessment of expected maintainability. A survey of reports from research and industry indicates that little emphasis is given on identifying the strategy that best fits specific measurement purposes. Although the specific purposes are special to every measurement initiative, this paper shows which decisions must be made while creating such a strategy, and suggests and discusses alternatives for each decision. The goal of the resulting strategy is to construct derived measures or views (through aggregation and combination) that can be interpreted so that specific questions can be answered with a certain level of confidence. In many cases and for many reasons, the historical data needed to create statistically based models are not available. Consequently, the interpretation of the derived measures needs to be partly based on judgments. This work promotes a systematic approach to the identification of alternative strategies for conducting system level assessment of maintainability. More empirical work is required to evaluate and improve the accuracy and precision of judgment-based strategies. A future scenario is to establish baselines of measurement values for specific software industry sectors, which could be used by software acquirers and providers as a basis for setting measurable goals on quality attributes that have previously been difficult to measure.

References

1. B. Boehm and H. In, "Identifying Quality-Requirement Conflicts," *IEEE Software*, vol. 13, pp. 25-35, 1996.
2. ISO/IEC, "Software engineering — Product quality — Part 1: Quality model," 2001.
3. S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476-493, 1994.
4. L. Briand and J. Wuest, "Empirical Studies of Quality Models in Object-Oriented Systems," *Advances in Computers*, vol. 59, pp. 97-166, 2002.
5. V. R. Basili, G. Caldiera, and H. D. Rombach, "Goal Question Metrics Paradigm," *Encyclopedia of Software Engineering*, vol. 1, pp. 528-532, 1994.
6. J. Bansiya and C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Transactions on Software Engineering*, vol. 28, pp. 4-17, 2002.
7. J. McCall, P. Richards, and G. Walters, "Factors in Software Quality," General Electric Command & Information Systems Technical Report 77CIS02 to Rome Air Development Center, Sunnyvale, CA 1977.
8. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, pp. 308-320, 1976.
9. M. H. Halstead, "Elements of Software Science, Operating, and Programming Systems Series," vol. 7, 1977.
10. P. Oman and J. Hagemeister, "Construction and Testing of Polynomials Predicting Software Maintainability," *Journal of Systems and Software*, vol. 24, pp. 251-266, 1994.
11. D. Tegarden, P., S. Sheetz, D., and D. Monarchi, E., "A software complexity model of object-oriented systems," *Decision Support Systems*, vol. 13, pp. 241-262, 1995.
12. M. Lorenz and J. Kidd, *Object-Oriented Software Metrics: A Practical Approach*: Prentice-Hall, 1994.

13. W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability," *Journal of Systems and Software*, vol. 23, pp. 111-122, 1993.
14. Y. S. Lee, B. S. Liang, S. F. Wu, and F. J. Wang, "Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow," presented at Conf. Software Quality, Maribor, Slovenia, 1995.
15. A. Lake and C. Cook, "Use of Factor Analysis to Develop OOP Software Complexity Metrics," presented at 6th Annual Oregon Workshop on Software Metrics, Silver Falls, Oregon, 1994.
16. M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion In Object-Oriented systems," presented at Int'l Symp. Applied Corporate Computing (ISACC '95), Monterrey, Mexico, 1995.
17. L. Briand, P. Devanbu, and W. Melo, "An Investigation into Coupling Measures for C++," in *Proceedings of the 19th international conference on Software engineering*. Boston, Massachusetts, United States: ACM Press, 1997.
18. D. Darcy and C. F. Kemerer, "OO Metrics in Practice," *IEEE Software*, vol. 22, pp. 17-19, 2005.
19. R. Subramanyam and M. S. Krishnan, "Empirical Analysis of CK metrics for Object-Oriented Design complexity: Implications for Software Defects," *IEEE Transactions on Software Engineering*, vol. 29, pp. 297-310, 2003.
20. F. e. Abreu, "The MOOD Metrics Set," presented at ECOOP'95 Workshop Metrics, 1995.
21. R. C. Sharble and S. S. Cohen, "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods," *SIGSOFT Software Engineering Notes*, vol. 18, pp. 60-73, 1993.
22. R. Harrison, L. G. Smaraweera, M. R. Dobie, and P. H. Lewis, "Comparing programming paradigms: an evaluation of functional and object-oriented programs," *Software Engineering Journal*, vol. 11, pp. 247-254, 1996.
23. T. Drake, "Measuring Software Quality: A Case Study," *Computer*, vol. 29, pp. 78-87, 1996.
24. J. Mayrand and F. Coallier, "System Acquisition Based on Software Product Assessment," presented at 18th International Conference on Software Engineering, Berlin, 1996.
25. R. Harrison, S. J. Counsell, and R. V. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *Software Engineering, IEEE Transactions on*, vol. 24, pp. 491-496, 1998.
26. J. Barnard, "A new reusability metric for object-oriented software," *Software Quality Journal*, vol. 7, pp. 35-50, 1998.
27. M. Schroeder, "A Practical Guide to Object-Oriented Metrics," *IT Professional*, vol. 1, pp. 30-36, 1999.
28. L. Briand and J. Wüst, "Integrating scenario-based and measurement-based software product assessment," *Journal of Systems and Software*, vol. 59, pp. 3-22, 2001.
29. M. Saboe, "The Use of Software Quality Metrics in the Materiel Release Process — Experience Report," presented at Second Asia-Pacific Conference on Quality Software, Hong Kong, 2001.
30. I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code quality analysis in open source software development," *Information Systems Journal*, vol. 12, pp. 43-60, 2002.
31. R. Ferenc, I. Siket, and T. Gyimothy, "Extracting Facts from Open Source Software," in *Proceedings of the 20th IEEE International Conference on Software Maintenance*: IEEE Computer Society, 2004.
32. R. C. Gronback, "Software Remodeling: Improving Design and Implementation Quality," Borland 2003.
33. T. Dybå, B. A. Kitchenham, and M. Jørgensen, "Evidence-based software engineering for practitioners," *IEEE Software*, vol. 22, pp. 58-65, 2005.
34. B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-based Software Engineering," presented at Proceedings of the 26th International Conference on Software Engineering (ICSE), Edinburgh, Scotland, 2004.
35. I. T. Jolliffe, *Principal Component Analysis*, 2nd ed. New York: Springer-Verlag, 2002.
36. "Telelogic Tau Logiscope 6.1 Audit – Basic Concepts." Malmö, Sweden: Telelogic AB, 2004.
37. M. Morisio, I. Stamelos, and A. Tsoukias, "A New Method to Evaluate Software Artifacts Against Predefined Profiles," in *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. Ischia, Italy: ACM Press, 2002.
38. J. Kontio, "A Case Study in Applying a Systematic Method for COTS Selection," presented at 18th International Conference on Software Engineering, Berlin, 1996.
39. L. Briand, C., J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems," *Journal of Systems and Software*, vol. 51, pp. 245-273, 2000.
40. K. D. Welker, P. W. Oman, and G. G. Atkinson, "Development and Application of an Automated Source Code Maintainability Index," *Journal of Software Maintenance: Research and Practice*, vol. 9, pp. 127-159, 1997.
41. J. Bieman, M. and B.-K. Kang, "Cohesion and Reuse in an Object-Oriented System," in *Proceedings of the 1995 Symposium on Software reusability*. Seattle, Washington, United States: ACM Press, 1995.
42. L. C. Briand and J. Wüst, "The Impact of Design Properties on Development Cost in Object-Oriented Systems," presented at Software Metrics Symposium, London, UK, 2001.