Using Concept Mapping for Maintainability Assessments

Aiko Fallas Yamashita
Simula Research Laboratory & Dept. of Informatics, University of Oslo, Norway
aiko@simula.no

Bente Anda
Norwegian Directorate of Taxes & Dept. of Informatics, University of Oslo, Norway
bentea@ifi.uio.no

Dag I. K. Sjøberg
Department of Informatics University of Oslo, Norway
dagsj@ifi.uio.no

Hans Christian Benestad
Simula Research Laboratory Lysaker, Norway
benestad@simula.no

Per Einar Arnstad
Scanmine A.S. Trollåsen, Norway
parnstad@scanmine.com

Leon Moonen
Simula Research Laboratory Lysaker, Norway
leon.moonen@computer.org

Abstract

Many important phenomena within software engineering are difficult to define and measure. One example is software maintainability, which has been the subject of considerable research and is believed to be a critical determinant of total software costs. We propose using concept mapping, a well-grounded method used in social research, to operationalize the concept of software maintainability according to a given goal and perspective in a concrete setting. We apply this method to describe four systems that were developed as part of an industrial multiple-case study. The outcome is a conceptual map that displays an arrangement of maintainability constructs, their interrelations, and corresponding measures. Our experience is that concept mapping (1) provides a structured way of combining static code analysis and expert judgment; (2) helps in the tailoring of the choice of measures to a particular system context; and (3) supports the mapping between software measures and aspects of software maintainability. As such, it constitutes a useful addition to existing frameworks for evaluating quality, such as ISO/IEC 9126 and GQM, and tools for static measurement of software code. Overall, concept mapping provides a systematic, structured, and repeatable method for developing constructs and measures, not only of maintainability, but also of software engineering phenomena in general.

1. Introduction

The maintainability of a software system is usually a critical determinant of software costs, yet it is very difficult to evaluate. A primary difficulty comes from the fact that software maintenance involves dealing with many factors, ranging from technological features to human dynamics and cognition, all of them comprising many different and complex settings. These contextual factors limit the generalization of the findings of individual studies. Hence, it is important to determine why the contextual factors play such an intrinsic role in maintenance. We consider, in line with Pizka & Deißenböck [1], that maintainability is not solely a property of a system, but touches on three different dimensions: (a) the people performing software maintenance, (b) the technical properties of the system under consideration, and (c) the maintenance goals and tasks.

With respect to dimensions (a) and (c), it is known that notions such as opportunistic comprehension [2] and information requirement [3] seem to explain some aspects of how large commercial software systems are understood and maintained. Such notions suggest that the skills and experience of the developer and the nature of the maintenance task drive the process of comprehending the system during maintenance. Yet procedures for maintainability assessment have paid little attention to how the programmer or maintainer understands the system or what their information needs are to perform a particular maintenance task. Many of the approaches to assessment tend to isolate the system from its environment by focusing only on the system's technical properties, which limits the scope and accuracy of these approaches.

With respect to dimension (b), most work suggests describing the technical properties of a system by quantifiable means, such as software measures, and connecting them afterwards to higher-level quality attributes. ISO 9126 [4] is an example of such an approach. Nowadays, many technical properties of the system can be measured automatically. Still, the central difficulty is to establish relationships between the quantifiable measures and the quality attributes, such as maintainability. One problem is that the effect of the technical properties on maintainability depends on the context of the system. Consequently, context-specific models are needed.

Given that the nature of the maintenance goals and tasks plays an important role during software maintenance, the

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

constructs representing software maintainability should, to some extent, be goal-driven. From a practical perspective, we should ask ourselves: *"Is the system good enough for what we plan to do with it?"* We conjecture that many inaccuracies generated by, and issues regarding construct validity with, software quality models are due to their rigid nature, which means that they cannot adapt to the specifics of the organizational context. That being so, there is a need to provide methodological support for building, adapting, and validating such models for a given context or setting. In this paper, we propose to use *concept mapping* [5] as a method to incorporate contextual information in the operationalization of software engineering constructs. We show how concept mapping can be used to systematically derive measures that can be analyzed and interpreted in order to assess the maintainability of a system. We suggest using expert judgment in the concept mapping process for deriving the contextual information. Anda [6] suggested that combining expert knowledge with static code analysis is a viable approach to evaluation, because these strategies address different attributes and dimensions of a system. Static code analysis enables the use of empirical evidence and existing models of software quality. Conversely, expert judgment incorporates contextual and cognitive factors into the analysis, thus supporting more realistic interpretations of the technical properties of the system.

The remainder of this paper is organized as follows. Section 2 describes the theoretical foundation of concept mapping. Section 3 describes how we used this method to develop a map of maintainability constructs according to a given perspective. Section 4 discusses the proposed method. Section 5 discusses some challenges that need to be met when using our approach. Section 6 summarizes the method and offers directions for further research.

2. Concept mapping

We now outline the theoretical foundations of concept mapping and, following Trochim [7], describe the steps required to implement it.

2.1 Programme evaluation

Concept mapping is a method commonly used in social research to plan and evaluate programmes [8]. Programme evaluation is a formalized approach for studying the goals, processes, and results of public and private projects, governmental development policies, and programmes. Programme evaluation and software assessments face similar challenges. For instance, programme evaluation models the aspects of the project, policy, or programme under evaluation. Similarly, in software engineering, a *quality model* of a given system should be defined on the basis of relevant socio-technical properties, such as, characteristics of the developers and privacy policies. While traditional software assessments focus mainly on the technical factors, we see great potential for approaches from social research to incorporate other factors as well. Concept mapping is one example of such an approach.

2.2 Concept mapping definition

To conceptualize a problem means to organize one's ideas about the topic of the problem such that pertinent entities, and the relations between them, are specified. This process first identifies the pertinent entities by acquiring input from several relevant theories or groups of people involved in the programme. Second, it determines the underlying relationships between the identified entities by using multidimensional scaling and cluster analysis. Concept mapping is one type of structured conceptualization. It consists of a sequence of concrete operational steps, which yields a *conceptual representation* [7] of the element(s) under analysis. Conceptualization processes are considered valuable for programme evaluation because they help to gather information about the actors in the project (various stakeholders, authority groups, etc.) from a variety of perspectives. Information about the various categories of participants represents one kind of context information of the programme.

2.3 The concept mapping process

As described by Trochim [5], a concept mapping process has six main steps: preparation of the process, generation of statements (in which the conceptual domain is defined), structuring of statements (the relationships between the domain entities are established), representation of statements (by textual, pictorial, or mathematical means), interpretation of concept maps, and utilization of concept maps. An *initiator* requests the concept mapping process, and a *facilitator* leads the group of *participants* through the various steps.

2.3.1. Preparation. The main goal of the preparation stage is to select the participants and define the focus of the conceptualization, as a precursor to generating statements about the conceptual domain. Trochim suggests selecting about 10 to 20 participants, although it is possible to use any number, from one to hundreds. After the participants have been selected, the initiator works with them to develop the focus of the conceptualization. Some examples of conceptualization focus for programme evaluation in are given in [7]: the nature of the policy, the desired outcomes, or the type of people to be included in the evaluation.

2.3.2. Generation of statements. Once the focus becomes clear, a set of statements within the focus is generated using brainstorming or alternative methods, such as brainwriting, nominal group techniques, focus groups, and qualitative text analysis (see also [9-11]). Trochim suggests that the number of statements should be kept to a manageable level.

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

According to Trochim's model for the conceptualization process, it is possible also to generate the statements one by one (e.g., by selecting from a predefined list of statements).

2.3.3. Structuring of statements. Once the statements are ready, they are printed onto cards and given to each participant. Then a technique called card sorting [12] is applied. Each participant is instructed to group the cards "in a way that makes sense to you". Trochim places several restrictions on this procedure: each statement can only be placed in one group (i.e., an item cannot be placed in two groups simultaneously); there must be at least two groups of statements; and, at least one group must have more than one statement. If the participants perceive that there are several different ways to group the cards, it is possible to have them select the most sensible arrangement, or to record several groupings for each participant.

When the grouping task is completed, the results are combined across people by using the following three steps:

a) The groupings made by each participant are recorded in a *binary symmetric similarity matrix* (see Figure 1).
b) The individual *similarity matrices* are combined into an *aggregated similarity matrix*. Each of the values in this matrix represents how many participants placed a given pair of statements together.
c) Each statement is rated according to the rating focus, using a five- or seven-item Likert.

	1	2	3	4	5
1	1	1	0	0	0
2	1	1	0	0	0
3	0	0	1	1	0
4	0	0	1	1	0
5	0	0	0	0	1

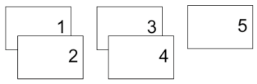

Figure 1: Example of a similarity matrix (left) for five statements grouped into three piles (right)

The aggregated similarity matrix provides information about how the participants grouped the statements; hence, it provides a representation of the relational structure of the conceptual domain. A high value implies that the statements are conceptually similar in some respect. A low value implies that they are conceptually more distinct. Both the generation and structuring of statements could involve individuals, groups, or the usage of non-subjective criteria, such as relevant theories or predefined algorithms (e.g., cluster analysis).

2.3.4. Representation of statements. In the representation stage, the grouping and rating input is represented pictorially by using two statistical analyses:

First, a *two-dimensional multidimensional scaling* [13] takes the grouping data across all participants and develops a *point map* in which each statement becomes a point on the map. The more people that have grouped the same statements together, the closer the statements are to each other on the map.

Second, a *hierarchical cluster analysis* [14] takes the output of the multidimensional scaling (the *point map*) and partitions the map into groups of statements or ideas, forming clusters. If the statements describe activities of a programme, the clusters show how these can be grouped into logical groups of activities. If the statements represent specific outcomes of a policy/programme, the clusters might be viewed as outcome constructs or concepts. This second map is called "*the conceptual domain of the outcomes*". Trochim suggests using Ward's hierarchical cluster analysis [7], [14], because it can generate several configurations of clusters, from which the participants can then decide which one makes the most sense.

Note that the statistical methods described above are not the only ones available for generating a cluster map. However, they are the most suitable for concept mapping. Trochim explains why in [7].

An additional result of the cluster analysis is the generation of a *cluster list* or a *named list of piles* [7] (into which the different statements are grouped), which is used in later stages of the concept mapping process.

Finally, each of the points can be assigned a given value, which can be represented by a bar. For example, the value may correspond to the perceived importance or criticality of a statement. The height of the bar then indicates the aggregated importance of the issue for the complete set of participants who rated the statements.

2.3.5. Interpretation of concept maps. The facilitator should work with the participants to help them to develop their own labels and interpretations for the different maps. In this analysis session, the participants use the cluster list to choose names for the different clusters, by negotiating (similarly to naming factors in factor analysis). The maps are then used to adjust the naming of the clusters and adjust the clusters themselves.

2.3.6. Utilization of concept maps. The maps can be used as a visual framework to implement or evaluate a given programme. They can also be used as the basis for developing measures and displaying results. Each cluster can be seen as a construct. The individual statements can suggest specific operationalizations of that construct.

2.3.7. Outputs. Several artefacts are created from the concept mapping process, for example, the statement list or the list of brainstormed statements, the cluster list, and the different maps (i.e., point map, cluster map, point-rating map). Another artefact is a cluster-rating map, which consists of the cluster map with average cluster ratings overlaid. This specific type of map was not included in this work. Examples of all these artefacts can be found in [7].

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

3. Towards a maintainability map

We now describe how we used concept mapping to generate a concept map for analyzing maintainability. We analyzed four web applications written in Java that manage information about empirical studies conducted by Simula Research Laboratory. All four systems conform to the same requirements specification but have considerable dissimilarities. For instance, their size varies from 7208 LOC for the smallest system to 14549 LOC for the largest one. Four Norwegian consultancy companies developed the four systems independently. More details of the case study that was conducted on these development projects can be found in [6], [15], [16].

Using Trochim's *concept mapping* approach, we derived conceptual maps, using as a basis software measures and other indicators relevant to the systems under study. This process resulted in a tailored two-dimensional point map that was composed on the basis of the measures, which were clustered into constructs that describe maintainability.

3.1 Preparation

The participants were four software engineering researchers and one professional software engineer. All had good knowledge of software maintenance in general and of the actual systems of analysis. The professional software engineer had more than 25 years of experience with software development. The researchers all had professional experience of software development, as well as good knowledge of code measures and design attributes.

The participants represented a convenience sample, in that all the researchers were at the time associated with the same research group (2nd, 3rd, 5th and 6th authors of this paper), and the engineer (4th author) had already evaluated the maintainability of the systems. The initiator was the 1st author. The focus question was: "Which characteristics could be used to better understand the maintainability of the four systems?"

3.2 Generation of statements

Our process for generating statements deviated slightly from Trochim's original approach. We were interested in exploring different software indicators and how useful for uncovering or predicting maintainability issues it would be to combine them in different ways. Instead of performing a brainstorming session, we preselected a set of *design attributes* as statements for our concept mapping. A *design attribute* is here widely defined as a code attribute, code smell, or design principle violation that is present in the design of a software system.

We did not perform a brainstorming session in this case because we wanted to include only code smells and design principle violations that were formally defined and detectable automatically. (The term *code smells,* which was coined by Kent Beck and Martin Fowler [17], is informally defined as bad or inconsistent parts of the design of object-oriented software.) We used Borland Together 2008 [18] for this purpose. In addition, we used the software measures suggested in [15]. These measures were the result of an analysis to find a minimal set of measures that could describe the dimensions of design that influence maintainability in these four systems [15].

Moreover, we included different software measures as part of the statements, so that we could use the knowledge derived from empirical studies on measures of software structural attributes [19-21]. Arisholm and Sjøberg [22] suggest that metrics may be more practical when used in combination than when interpreted individually.

However, from a practical point of view, analysis that relies only on software measures does not offer clear guidance to developers about how to improve maintainability. To overcome this limitation, we incorporated code smells and design principle violations (defined as *structural symptoms*) to complement the analysis of software measures.

Our motivation for integrating structural symptoms into the list of statements is that for each of these, redesign strategies (e.g., refactoring, use of design patterns) are available for improving the software [17]. There has been a growing interest in the topic of code smells for assessments of software maintainability. Van Emden and Moonen [23] provided the first formalization of code smells and described a tool that could detect them. Marinescu [24] further formalized the definition of code smells and extended the detection to a wider range of code smells and a number of design principle violations.

The resulting list of design attributes used as statements is presented in Table 1. It shows for each statement, its number, the name of the corresponding design attribute and the type of design attribute, where 'CS' means Code Smells, 'DPV' means Design Principle Violations, and 'CM' means Code Measures.

3.3 Structuring of statements

The statements were structured in two steps. The first step consisted of a discussion and elicitation session. The second step consisted of grouping the statements. The main purpose of the discussion and elicitation session was to discuss the implications of the selected software design attributes on the systems, and how they might be interrelated or grouped together according to different perspectives. Given that the perspectives used for the grouping could be diverse, the participants were required to give reasons for grouping the statements in the way they did. For instance, some people might relate or group measures using base-rate, relationship, or causality viewpoints, whereas others may relate two attributes using a risk management viewpoint (e.g., "grouping the most risky ones together").

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

Table 1: List of statements (design attributes)

No.	Design Attribute	Type	No.	Design Attribute	Type
1	Interface segregation principle (ISP) violation	DPV	18	Refused bequest	CS
2	Data class	CS	19	Subclasses have the same member	CS
3	God method	CS	20	Feature envy	CS
4	Tight class cohesion (TCC)	CM	21	Suspicious usage of switch statements	CS
5	Temporary variable is used for several purposes	CS	22	Import list construction	CS
6	Comments: Lines of comments in the code	CM	23	Field is used as a temporary variable	CS
7	Usage of implementation instead of interface	DPV	24	Number of children (NOC)	CM
8	Shotgun surgery	CS	25	Unused local variable or formal parameter	CS
9	Call from methods in an unrelated class (OMMEC)	CM	26	Duplicated code in constructors	CS
10	Depth of inheritance tree (DIT)	CM	27	Member is not used	CS
11	Data clump	CS	28	Call to methods in an unrelated class (OMMIC)	CM
12	Long message chain	CS	29	God class	CS
13	Number of lines of code (LOC)	CM	30	Declaration is hidden	CS
14	Single responsibility principle	DPV	31	Dead code	CS
15	Unused class	CS	32	Wide subsystem interface (lack of façade)	DPV
16	Duplicated code in conditional branches	CS	33	Number of methods in a class (WMC)	CM
17	Misplaced class	CS			

All these perspectives were considered to be valid, given that our goal was to find the perspectives that were most relevant for the evaluation of the systems according to the type of system and type of maintenance tasks that were planned for them.

The first step started with an explanation of the session. The list of design attributes was then presented to the participants. For each of the attributes, a discussion was held about: (a) what effects it might have on the system, (b) what the reason was behind their presence (their nature), (c) how they were related to other attributes from a certain point of view, and (d) whether any other perspectives could be applied.

The facilitator compiled the comments from the participants. After the session, the participants were assumed to have acquired enough contextual information, as well as a reasonable perception of how these different measures and smells could be grouped. For the second step, the participants were requested to group the set of design attributes by using a web tool [25].

3.4 Representation of statements

The purpose of this session was to obtain group agreement on the aspects (which were represented in the names of the clusters) of maintainability that should be used in order to interpret the Code Smells, Design Principle Violations, and Code Measures. The outcome of the session should be a point map and a cluster map, composed of a set of design attributes that represents different aspects of the software design.

Given that the participants made several groupings according to different perspectives, a second session was needed in order to synchronize the participants' interpretation of the perspectives and the names of the clusters. Normally, only one perspective is used for concept mapping, but we initially allowed several perspectives (or several groupings, each using a different perspective) to enable the one(s) that is most important for the assessment to be selected.

During the session, the *cluster lists* for the participants were presented and discussed, in order to group the different cluster names together according to a given perspective. For instance, one of the chosen perspectives of two of the participants was the "Severity" of the design attributes. One of the two used "Severe", "Moderate", "Dependent of Context" and "Low" as cluster names for grouping the statements. The other used "Serious", "Suspicious", "Unimportant" and "Need a balance" as cluster names.

In cases in which the perspective was not sufficiently clear, the sets of individual measures that belonged to the clusters were compared in order to see if they were similar, in which case it was assumed that the clusters represented similar concepts. Table 2 lists the perspectives that were generated after the discussion. We focused on "Design/Structural issues" because that allowed us to compare the results of our approach with other available quality models [26] that share the same perspective. We aggregated the groupings that used the "Design/Structural issues" perspective using a tool [25] that implements both multidimensional scaling and hierarchical cluster analysis. The resulting point map is shown in Figure 2. The outcome from the hierarchical cluster analysis (a cluster map) is shown in Figure 3.

As part of the participants' grouping of the statements in piles (see Section 3.3), they were asked to come up with a representative name for each of the piles. The cluster analysis algorithm selects a name for each of the clusters on the basis of the names proposed initially by the participants to designate each of the piles. The naming is adjusted accordingly during the negotiation stage, which is described in Section 2.3.5.

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

Table 2: List of perspectives drawn from discussion

Perspective	Description
Severity	The level of risk they might represent (e.g., from "Severe" to "Low")
Predictability	The level of uncertainty they represent (also following a risk-assessment perspective)
Properties of developers	Human factors that indicate the type of developers who implemented the system
Design/Structural	Software design concepts
Abstraction level	Level of responsibility for the required refactoring or redesign (e.g., "Programmer's responsibility", "Designer's responsibility" and "Architect's responsibility")
Cause indicators	Representing potential events/factors that lead the system to display these attributes (e.g., "Bad coding practices", "Design unused or used for informal testing")

3.5 Interpretation of concept maps

The remainder of the session aimed at adjusting the cluster map. Clusters representing similar concepts and measures were identified (see Columns 1 and 2 in Table 3), and names for common concepts were agreed upon. The cluster map that resulted from the discussion is shown in Figure 4.

In some cases, it was not possible to find an intersection of all the conceptually equivalent piles. For instance, in cluster 7 "Decomposition", there are common attributes between clusters of the participants P1, P3, and P4, but the intersection would be empty if the cluster of participant P2 were considered. In those cases, the nearness of the attributes in the point map or in the previously generated cluster map was used as a basis for deciding which attributes to include in the final cluster.

In cases where there was no final agreement on the cluster to which an attribute should belong, the issue was left open, as in the case of cluster 4 "Duplicates" and cluster 10 "Structural problems", which share Attribute 11 (Figure 4).

3.6 Utilization of concept maps

We retrieved the values for the measures, using Borland Together 2008 [18]. To compare measures on different scales, the measures were standardized across the values from each of the systems. Due to limitations of space, we present only two examples of very contrasting systems: A and B (see Figure 5 and Figure 6). Due to the fact that the measures within the clusters do not have a common measurement unit (e.g., clusters can contain smells, measures, and design violations), using a cluster-rating map would not be adequate for our analysis. A cluster-rating map does not allow for the interpretation of individual indicators. In addition, displaying a mean value of measures with different units would make it difficult to interpret the mean values correctly.

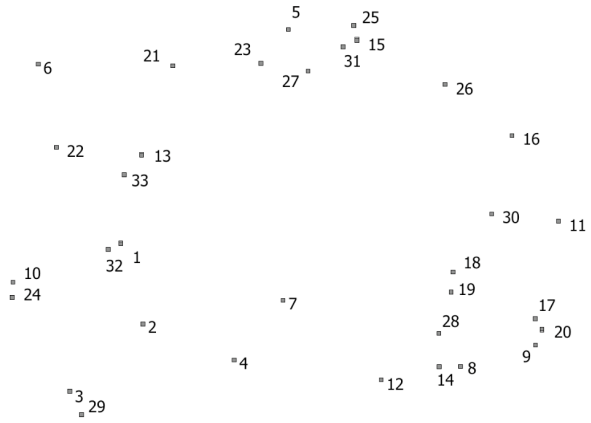

Figure 2: Point map derived by our concept mapping

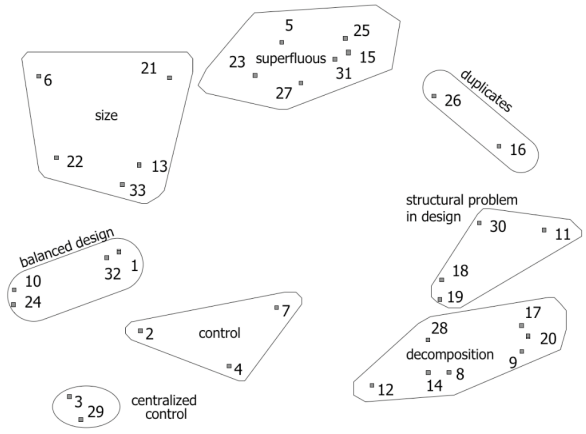

Figure 3: Cluster map generated by Concept System

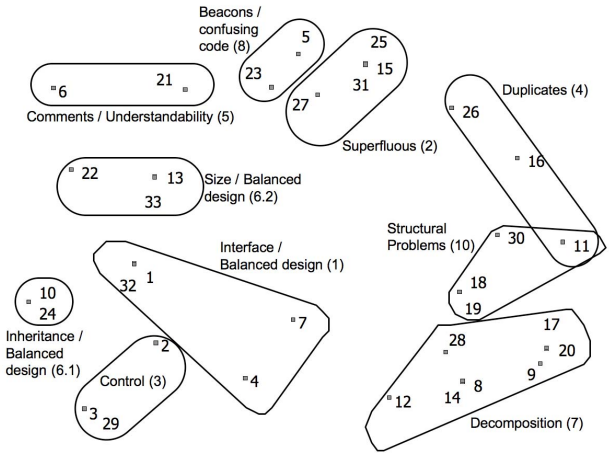

Figure 4: Final cluster map after discussion

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

Table 3: Cluster names chosen for distinguishing the conceptually equivalent clusters, the statements per participant per cluster, and the final statements of the cluster after the negotiation stage

Cluster name	Partici-pant	Statements of the equivalent piles	Statements
Interface / Balanced design	P1	1, 7, 30, 32	1, 4, 7, 32
	P2	1, 7, 22, 27, 30, 32	
	P3	1, 2, 4, 7, 22, 26, 32	
Superfluous	P1	15,25,27,31	15, 25, 27, 31
	P4	15,16,21,22,25,26, 27	
	P5	15,18,25,27,31	
Control	P1	3,12,23,29	2, 3, 29
	P4	3,6,29	
	P2	2,3,4,12,14,29	
	P3	3,10,24,29	
Duplicates	P1	2,5,11,16,26	11, 16, 26
Comments / Understandability	P1	6,21,22	6, 21
Inheritance / Balanced design	P1	4,10,13,24,28,33	10, 24
	P4	1,2,10,13,18,19,24, 32,33	
Size / Balanced design	P1	4,10,13,24,28,33	13, 22, 33
	P4	1,2,10,13,18,19,24, 32,33	
Decomposition	P1	8,9,14,17,18,19,20	8, 9,12, 14,17,20, 28
	P4	4,7,8,9,11,12,14,17, 20,28	
	P2	10,18,19,21,24	
	P3	1,2,4,7,22,26,32	
Beacons / Confusing code	P4	5,23,30	5, 23
	P3	6,9,13,33	
	P2	5,6,13,15,21,23,25, 27,31,33	
Structural problems	P5	9,10,17,19,20,21,22,26	11, 18, 19, 30
	P3	8,9,11,12,14,16,17, 18,19,20,28,30	

Instead of using a cluster-rating map, we used a combination between a cluster-map and point-rating map, in order to compare the individual measurement values across the different clusters. We represent the relative difference between the values in the form of a 10-scale value (with the smallest value found being -1.4 (one block) and highest value being 1.5 (nine blocks)) and assign five blocks to the closest values to the mean.[1] The motivation for using a 10-scale bar was that it gives us the right level of granularity for visualizing the differences between the systems.

We used the four maps to analyze the differences between the four systems. Concept maps may also be used to generate hypothetical patterns in pattern matching [27]. Pattern matching requires drawing a theoretical pattern of expected outcomes (comparable to a hypothesis) on the basis of a pattern of characteristics of a given object or phenomenon (similar to control variables). While experimental studies typically involve univariate analysis, pattern matching follows the multivariate analysis perspective, which is very suited to case studies. See our plans with respect to pattern matching in Section 5.

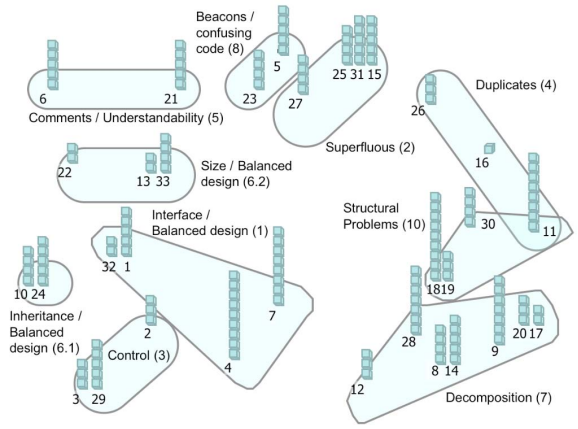

Figure 5: A cluster map with measurements from System A

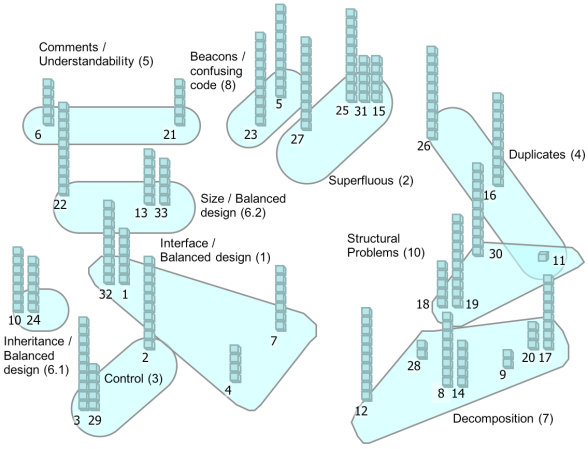

Figure 6: A cluster map with measurements from System B

3.7 Outcome and general remarks

The findings from the mapping process and subsequent analysis will be presented in the following subsections.

3.7.1. Structuring of expert knowledge. To exemplify how expert knowledge in this case was structured and operationalized through concept mapping, we compared the results from the expert evaluation reported in [6] and the resulting concept maps. One example is the comments from the expert about System B using a comprehensive proprietary library. This may be viewed as relating to cluster 6.2 *Size/Balanced design* (measure 22: *Import list construction*). This cluster also indicates considerable differences in their size (A has 7937 LOC vs. B with 14549 LOC).

[1] From the set of standardized values of the measures across the 4 systems, -1.4 was approximately the smallest value and 1.5 was the highest value.

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

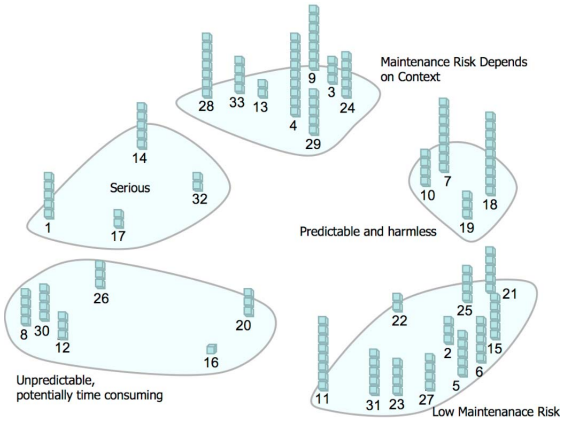

Figure 7: A cluster map with measurements from System A using the Severity/Predictability perspectives

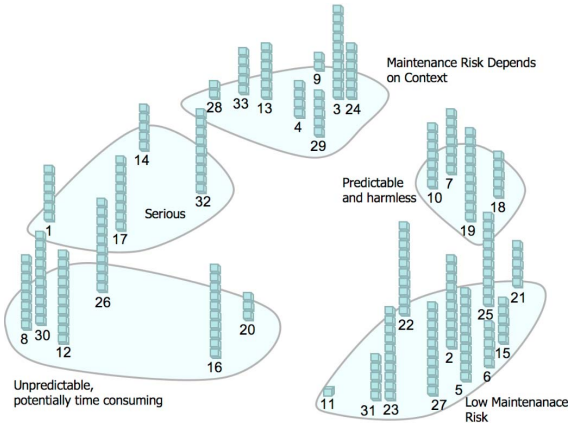

Figure 8: A cluster map with measurements from System B using the Severity/Predictability perspectives

Cluster 3 *Control* has a considerable difference between the systems for Measure 2: *Data Class*. In addition, in the *Control* cluster, system B has many more God Methods than System A, which could be related to the factors to which the experts referred in *Choice of classes* from [6]:

A: Contains primary objects that are implemented with classes that contain both data and logic.

B: Has primary objects, which are implemented as containers, and they also remark that there are additional, unnecessary containers.

Another example is the description by the experts regarding *Inheritance*, where they stated:

A: Mostly successful use of inheritance, but in some cases the base class does not contain all the functionality that is expected in a base class.

B: Too extensive use of inheritance. Confusions regarding whether functionality should be in the base class or the subclass.

This difference between the systems can be seen in Cluster 6.1: *Inheritance/Balanced design* where the values for Measures 10 (*Depth of Inheritance Tree)* and 24 (*Number of Children)* are considerably higher in B.

In addition, the use of objective observations (in this case software measures) helps to reduce misinterpretations, which are often the result of evaluations described informally in unstructured text.

3.7.2. Interpretation from different perspectives. The perspective chosen by most of the researchers was different from the one chosen by the professional software engineer. Where the prevalent perspective of the researchers was similar to the taxonomy by Mantyla [28], the dimensions used by the software engineer related more to the predictability levels of each of the attributes, as well as the potential severity they represent (i.e., they adopted what may be termed a risk analysis perspective).[2]

From the discussion, it became clear that these two perspectives were strongly intertwined. That being so, we repeated the process, this time choosing a combination between Severity and Predictability perspectives.

The result is shown in Figure 7 and Figure 8, where the clusters have more or less two major segments, one side with the unpredictable/serious indicators and the other side with the measures deemed not so serious.

An interesting aspect is that the cluster *Maintenance Risk Depends on Context* is composed mainly of software measures and code smells that are related to size and control (e.g., *God Class*, *God Method*), which also reflects the need for additional detail in order to categorize these indicators as harmful or not. A God Class may appear to be the only option for architectures that require centralized control. However, of course, this is not clear from just looking at the characteristics of the system. Therefore, an additional map that describes the context of the project (which could state the nature of the maintenance tasks, the developers' skills, and the requirements of the system) would be needed to evaluate these indicators.

Although systems A and B have relatively similar values in the cluster *Maintenance Risk Depends on Context* (which might indicate that either systems could 'behave nicely' depending of the setting), the high values from system B on the clusters *Serious* and *Potentially time consuming* should raise a flag to the evaluators if they want to consider system B for a project with very strict deadlines. Due to limitations of space, we are unable to present details regarding the results from this analysis.

4. Benefits of Concept Mapping

We discuss four subareas of software assessment in which concept mapping may improve on current

[2] The severity perspective was also chosen by two of the researchers.

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

approaches, state some of the challenges that it faces, and offer recommendations for using this method in practice.

4.1 Transparent quality models

Numerous models and frameworks have been proposed for evaluating software quality in industrial settings. For instance, in the *Goal Question Metric* [29] paradigm, the measurement models are derived by linking measurement goals to operational questions, which can be answered by measuring aspects of products, processes, or resources.

Other models follow the *hierarchical approach* [30-32] and breakdown external quality attributes into internal attributes and from there to low-level measures.

Examples of other approaches to quality models that follow the *Factor Criteria Metric* paradigm [33] are the complexity model for object-oriented systems by Tegarden et al. [34] and the Maintainability Index (MI) by Oman and Hagemeister [32], which is typically used for evaluating maintainability at system level.

However, as Marinescu points out [35], many of these approaches have an implicit mapping between observable measures and the abstract attributes; hence, the criteria that are used for the mapping are not made explicit. In many cases, neither the criteria nor the process followed for deriving these criteria are stated. Part of the focus with these models has been on establishing empirical validation of the mapping criteria, but again, many studies have limited generalizability due to the lack of contextual information and the inherent complexity of the projects.

Concept mapping helps to provide explicit descriptions of these mapping criteria and the process by which they are derived. The mapping criteria can later be adapted and improved, using empirical evidence and expert knowledge as a basis. Another advantage with concept mapping is that it provides a pictorial representation, where the measures within the attributes are not concealed. This allows us to observe the measures in relation to all other measures, and to see how each of them fit into the overall picture. In our case, we found that multivariate representations support better data comparison across systems and thus facilitate data interpretation. For instance, if we compare Kiviat charts to concept maps, we find that the latter are more scalable in terms of number of displayable measures, thereby providing better use of the 2D space for representing the relations between the different measures.

4.2 Developing tailored quality models

The ISO/IEC 9126 standard [4] outlines software quality attributes and decomposes them into subattributes in a hierarchical way. While this can be a good starting point, the operationalizations of the attributes are specified only partially. Concept mapping could guide the operationalization by starting from low-level properties (i.e., the statements) and inferring their meaning according to a given goal or purpose. For example, if we intended to undertake adaptive maintenance, we could group the measures according to such aspects as interfaces, separation of concerns, and level of definition of architectural layers, and link them back to the higher-level attribute of *adaptability*. Another example of how to use concept mapping is to start from the quality attributes of ISO (five in addition to maintainability) and use them as focus topics. Groups of experts could then generate statements that operationalize the five high-level attributes, as we have done for maintainability.

In the same way as ISO, ontologies such as the one by Kitchenham et al. [36] identify and describe a series of factors that were thought to affect maintenance and empirical studies of maintenance. Kitchenham's ontology outlines a wide range of concepts, and it is still a challenge to identify which are the relevant ones for a given purpose. More specialized ontologies have been developed, for example, for describing software maintenance and evolution [37] and for performing software design analysis [28]. This illustrates the need to adapt quality models into different domains of concern [26]. Concept mapping can help the construction of domain- and context-specific ontologies that is based on the knowledge and experience of experts. In a given area, domain experts may have other indicators that are more comprehensive than the ones in the standard quality models (See Li and Smidts [38]). Concept mapping could be useful for identifying the indicators that the experts use. Given that concept mapping requires that a focus be chosen for the generation and structuring of the statements, it will guide the selection of relevant aspects in order to derive a framework for assessment, ontology, or quality model.

4.3 Representing contextual information

Throughout the paper, we have emphasized the importance of contextual information for assessment purposes. Mayrand and Coallier [39] exemplify how to incorporate contextual information by describing a process-oriented procurement project that combines capability assessment with static analysis. The authors point out that the lack of contextual description makes it difficult to interpret the metrics. For instance, from a metric point of view, a decision tree may seem complex, but from the programmer's point of view, the logic is well organized and easy to understand and validate.

Briand and Wüst [40], and Bengtsson and Bosch [41] have incorporated contextual information by integrating *change scenarios* with software measures. Correspondingly, Van Deursen and Kuipers [42] propose a method of risk assessment that is based on software product and process analysis.

Concept mapping can be used to generate representations, not only of systems, but also of different

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

projects. The latter category of maps might be equivalent to Trochim's *programme-characteristic pattern* [43]. Having a visual representation of contextual data will enable the decision-makers to consider the different factors involved in the project and consequently interpret the relative importance of the different characteristics of the system. Carr and Wagner [44] report that experts consider contextual aspects, such as the cost, urgency, and difficulty of a maintenance task, while making decisions concerning how a given maintenance goal or need is, respectively, to be achieved or met.

4.4 Incorporating expert knowledge

We have proposed the use of expert judgment in conjunction with concept mapping. Expert judgment in software engineering has been used for management and decision-making purposes [45], [46]. Moreover, Carr and Wagner [44] reported that experts use case-based reasoning and heuristics to prioritize maintenance projects. The heuristics that experts use are the result of knowledge gained from experience of past projects, in which different problems were solved with different solutions. As we noted in Section 0, concept mapping can help to structure and accumulate expert knowledge.

Heitlager et al. [47] stresses the need for cost-effective measurement frameworks that support root-cause analysis to identify specific problems and connect them to specific solutions. One reason why *Design Patterns* and *Code Smells/Refactorings* have gain popularity, is that they map a solution to a specific problem [17]. At a higher level of abstraction, concept mapping can be used to develop pattern-based evaluations, following the same line of thought as design patterns. By using the opinions of experts as input, we can use concept mapping to generate representations of both project contexts and the objects under evaluation. This could guide the process of defining the most desirable characteristics in a system for particular contexts.

5. Challenges and limitations

The main limitation of concept mapping is that any method that uses experts relies on the availability of sufficiently qualified experts. It might be thought that diverging opinions or lack of knowledge among the participants would be a problem. However, in a practical setting, the participants should be system stakeholders. That being so, strong divergence of opinion or a lack of the qualifications necessary to make good judgments about the system's maintainability might be a good sign that there are problems inherent in the project; hence, divergence of opinion and lack of knowledge do not necessarily constitute a limitation of concept mapping.

Although concept mapping is not particularly time-consuming, the use of this method does generate some overhead, which needs to be weighted alongside the ISO standard or GQM. However, note that even when using the well-established methodologies, interpreting metrics might take considerable time.

We are aware that in some situations, a simple model might be better suited to solving the problem than concept mapping. However, when we introduce concept mapping into the software engineering discipline, it is under the assumption that there are many similarities between maintainability assessments and programme evaluation, and in the programme evaluation domain it is evident that a simple model does not suffice for evaluation purposes.

A limitation in the concept mapping process that we report herein was the low number of participants (5) compared with the number prescribed by Trochim.

In addition, we are aware that a prescribed set of statements (the design attributes) could result in a loss of important contextual information for the evaluation. In order to address this limitation, an additional focus could have been to identify the contextual aspects that were relevant to the maintenance project (e.g., the type of maintenance tasks) so that these could have been considered explicitly during the discussion/generation/grouping of the statements.

6. Conclusions and future work

The software engineering industry has a strong need for methods to assess software maintainability that go beyond what evaluation purely based on Code Measures can provide. We find that concept mapping constitutes a strong addition to existing frameworks for evaluating quality, such as ISO/IEC 9126 and GQM, and tools for static measurement of software code. Our overall experience is that concept mapping provides a systematic, structured, and repeatable method for developing constructs and measures of the phenomenon of interest. It is useful for defining constructs and measures of other aspects of software engineering, in addition to maintainability.

Some areas for future work pertain to deriving an efficient and reliable method for providing human assessment in software maintainability using concept mapping. The model needs to be efficient if it is to gain acceptance in the industry, because the time spent on using this approach will be weighed against the gain in efficiency and risk assessment in the overall maintenance project. Reliability and accuracy can be improved by ensuring a consistent interpretation of the grouping activities and subsequent negotiating process, thereby reducing the chance for misinterpretation and misunderstanding. Incorporating repeated expert assessments of various maintenance tasks might provide a valuable general knowledge base that can be used in combination with specific contextual input in the maintenance project to be evaluated. This would make it possible to extend tools that

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

rely on evaluation of Code Measures to include context-specific assessment of software maintainability.

Our most immediate plan is to use pattern matching to analyze the data from a multiple-case study in which several maintenance tasks were performed on each of four systems (two of which are described in this article). In this case, the descriptive maps of the systems are the *control variables*. Since we need to generate a hypothetical pattern, another concept mapping session was conducted to generate a pattern for the outcomes of maintenance projects (called *process outcome pattern* by Trochim [43]). This pattern comprises different aspects that could describe the outcomes from a given project, in our case representing the outcomes from the multiple-case study. We drew four different outcome patterns, using our assumptions for the results from each of the four systems maintenance projects. We expect that this type of data analysis will prove useful for determining the usefulness of different measures for predicting outcomes from maintenance projects.

References

[1] Pizka, M. and F. Deissenboeck. How to effectively define and measure maintainability. in Softw. Measurement European Forum. 2007. Rome, Italy.

[2] Stanley, L., Cognitive processes in program comprehension, in Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers. 1986, Ablex Publishing Corp.: Washington, D.C., United States.

[3] O'Brien, M., Software comprehension: A review and research direction. 2003, University of Limerick.

[4] ISO/IEC, International Standard ISO/IEC 9126, International Organization for Standardization. 1991, Geneva.

[5] Trochim, W.M.K. and R. Linton, Conceptualization for planning and evaluation. Evaluation and Program Planning, 1986. **9**(4): p. 289-308.

[6] Anda, B. Assessing Software System Maintainability using Structural Measures and Expert Assessments. in Intl. Conf. on Softw. Maint. 2007.

[7] Trochim, W.M.K., An introduction to concept mapping for planning and evaluation. Evaluation and Program Planning, 1989. **12**(1): p. 1-16.

[8] Rossi, P.H., M.W. Lipsey, and H.E. Freeman, Evaluation: A Systematic Approach. 2004, Thousand Oaks, CA: SAGE.

[9] VandeVen, A.H. and A.L. Delbecq, The Effectiveness of Nominal, Delphi, and Interacting Group Decision Making Processes. The Academy of Management Journal, 1974. **17**(4): p. 605-621.

[10] Mary, C.L. and A.J. Marius, Understanding qualitative data: a framework of text analysis methods. J. Manage. Inf. Syst., 1994. **11**(2): p. 137-155.

[11] Morgan, D.L., Focus Groups as Qualitative Research, ed. S. Publications. 1988, Newbury Park, CA.

[12] Rosenberg, S. and M.P. Kim, The Method of Sorting as a Data-Gathering Procedure in Multivariate Research. Multivariate Behavioral Research, 1975. **10**(4): p. 489 - 502.

[13] Kruskal, J.B. and M. Wish, Multidimensional Scaling. Sage University Paper series on Quantitative Applications in the Social Sciences. 1978, Newbury Park, CA: Sage Publications.

[14] Everitt, B., Cluster analysis. 2nd Edition ed. 1980, New York: Halsted Press, A Division of John Wiley and Sons.

[15] Benestad, H., B. Anda, and E. Arisholm, Assessing Software Product Maintainability Based on Class-Level Structural Measures, in Product-Focused Software Process Improvement. 2006. p. 94-111.

[16] Anda, B.C.D., D.I.K. Sjoberg, and A. Mockus, Variability and Reproducibility in Software Engineering: A Study of Four Companies that Developed the Same System. Software Engineering, IEEE Transactions on, 2009. **35**(3): p. 407-429.

[17] Borland. Together. 2008 [cited 2008 September]; Available from: http://www.borland.com/us/products/together.

[18] Kafura, D. and G.R. Reddy, The Use of Software Complexity Metrics in Software Maintenance. IEEE Trans. Softw. Eng., 1987. **13**(3): p. 335-343.

[19] Wei, L. and H. Sallie, Object-oriented metrics that predict maintainability. J. Syst. Softw., 1993. **23**(2): p. 111-122.

[20] Genero Bocco, M., D.L. Moody, and M. Piattini, Assessing the capability of internal metrics as early indicators of maintenance effort through experimentation. J. Softw. Maint. and Evolution, 2005. **17**(3): p. 225-246.

[21] Arisholm, E. and D.I.K. Sjoberg, Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. IEEE Trans. on Softw. Eng. , 2004. **30**(8): p. 521-534.

[22] Fowler, M., Refactoring: Improving the Design of Existing Code. Refactoring: Improving the Design of Existing Code. 1999: Addison-Wesley.

[23] Van Emden, E. and K. Moonen, Java quality assurance by detecting code smells, in Working Conf. on Reverse Engineering. 2002. p. 97-106.

[24] Marinescu, R. Measurement and quality in object-oriented design. in Intl. Conf. on Softw. Maint. 2005.

[25] Concept Systems. 2009 [Dec. 2008]; Available from: http://www.conceptsystems.com.

[26] Kitchenham, B., et al., The SQUID approach to defining a quality model. Softw. Quality J., 1997. **6**(3): p. 211-233.

[27] Yin, R., Case Study Research : Design and Methods (Applied Social Research Methods). 2002: SAGE.

[28] Mantyla, M., J. Vanhanen, and C. Lassenius, A Taxonomy and an Initial Empirical Study of Bad Smells in Code, in Intl. Conf. on Softw. Maint. 2003, IEEE Computer Society.

[29] Basili, V.R., G. Caldiera, and H.D. Rombach, Goal Question Metrics Paradigm, in Encyclopedia of Software Engineering. 1994. p. 528-532.

[30] ISO/IEC, Software Engineering - Product quality - Part 1: Quality model. 2001.

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

[31] Jagdish, B. and G.D. Carl, A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Trans. Softw. Eng., 2002. **28**(1): p. 4-17.

[32] Oman, P. and J. Hagemeister. Metrics for assessing a software system's maintainability. in Intl. Conf. on Softw. Maint. 1992.

[33] McCall, J.A., P.G. Richards, and G.F. Walters, Factors in Software Quality Vol. I. 1977, Springfield, VA: NTIS.

[34] Tegarden, D.P., S.D. Sheetz, and D.E. Monarchi, A software complexity model of object-oriented systems. Decis. Support Syst., 1995. **13**(3-4): p. 241-262.

[35] Marinescu, R., Measurement and Quality in Object Oriented Design, in Department of Computer Science. 2002, "Politehnica" University of Timisoara.

[36] Kitchenham, B.A., et al., Towards an Ontology of software maintenance. Journal of Software Maintenance, 1999. **11**(6): p. 365-389.

[37] Christoph, K., B. Abraham, and T. Jonas, Mining Software Repositories with iSPAROL and a Software Evolution Ontology, in Intl. Workshop on Mining Softw. Repositories. 2007, IEEE Computer Society.

[38] Li, M. and C. Smidts, A Ranking of Software Engineering Measures Based on Expert Opinion. IEEE Trans. Softw. Eng., 2003. **29**(9): p. 811-824.

[39] Mayrand, J. and F. Coallier, System acquisition based on software product assessment, in Intl. Conf. on Softw. Eng. 1996, IEEE Computer Society: Berlin, Germany.

[40] Briand, L.C. and J. Wust, Integrating scenario-based and measurement-based software product assessment. J. Syst. Softw., 2001. **59**(1): p. 3-22.

[41] Bengtsson, P. and J. Bosch, An experiment on creating scenario profiles for software change. Ann. Softw. Eng., 2000. **9**(1-4): p. 59-78.

[42] vanDeursen, A. and T. Kuipers, Source-Based Software Risk Assessment, in Proceedings of the International Conference on Software Maintenance. 2003, IEEE Computer Society.

[43] Trochim, W.M.K., Outcome pattern matching and program theory. Evaluation and Program Planning, 1989. **12**(4): p. 355-366.

[44] Mahil, C. and W. Christian, A Study of Reasoning Processes in Software Maintenance Management. Inf. Technol. and Management, 2002. **3**(1-2): p. 181-203.

[45] Jørgensen, M., A review of studies on expert estimation of software development effort. Journal of Systems and Software, 2004. **70**(1-2): p. 37-60.

[46] Rosqvist, T., M. Koskela, and H. Harju, Software Quality Evaluation Based on Expert Judgement. Software Quality Control, 2003. **11**(1): p. 39-55.

[47] Heitlager, I., T. Kuipers, and J. Visser. A Practical Model for Measuring Maintainability. in Intl. Conf. on Quality of Information and Communication Technology. 2007

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