

Requirements on Maintainability of Software Systems - An Investigation of the State of the Practice

Jens Gustavsson
Linköpings universitet
Dept. of Computer and Information Science
SE-581 83 Linköping, Sweden
+46 13 282601

jens.gustavsson@ida.liu.se

Magnus Österlund
Ida Infront AB
Box 576
SE-581 07 Linköping, Sweden
+46 13 373764

magnus.osterlund@idainfront.se

ABSTRACT

This paper describes a study of how common requirements on maintainability are and how such requirements are expressed in a set of 10 real requirement specifications for software systems. To facilitate the study we propose a categorization of entities on which it can be useful to have maintainability requirements. There is a widespread interest in getting maintainable systems. However, there is much room for improvement in how maintainability requirements are made. Only a small fraction of the possible categories of maintainability requirements is utilized and the requirements that do exist in the specifications are generally poorly specified. We performed a literature study that shows that there is no clear consensus on how maintainability requirements should be expressed. Not many of the proposed ideas from literature are widely used in practice.

Keywords

Maintainability, requirements, requirement specifications, case-study

1. INTRODUCTION

Software maintenance is the activity of correcting faults in the software, adapting it to changes in its environment and changing the functionality of it. Maintenance consumes between 40 and 80 percent of software costs and is therefore probably the most important life cycle phase of software [10, 11, 13].

The large amount of maintenance has sometimes been considered a problem and there has been an ambition to reduce it. However, 60 percent of the maintenance costs come from making enhancements, which is something that make the systems provide additional value [11, 13]. It has also been shown [8] that systems built with modern development methods, such as structured analysis, prototyping and computer-aided software engineering (CASE), are more reliable than others, but still require more maintenance. This maintenance is less focused on emergency fixing and more on functional enhancements.

Robert Glass argues that maintenance should not be seen as a problem, but as a solution to how to build something only a bit different from what we have built already. Therefore, the goal should not be to have maintenance diminished [11]. Rather, it is reasonable to expect that maintenance will be as important or even more important in the future.

A software system quality attribute that is closely related to maintenance is *maintainability*, i.e. how easy it is to perform maintenance on the system. High maintainability is important, since it is a way to lower the cost for maintenance activities and thereby making it possible to lower the total maintenance cost or to make the system better by performing more maintenance activities at the same total cost. We believe that it is important to address the maintainability quality when creating new software systems. A reasonable first step in doing so would be to place requirements on maintainability in the requirement specification.

In this paper we describe our study on the state of the practice regarding maintainability requirements. We have investigated requirement specifications from a number of Swedish official authorities to find out how requirements on maintainability are handled in practice, in the domain of public procurement. The following research questions have been addressed by our investigation:

- How common are requirements on maintainability in practice?
- What kinds of requirements on maintainability are common and how are they expressed?
- How well does the practice of maintainability requirements correspond with theories described in literature?

In order to answer the second question we have created a classification scheme for maintainability requirements. In this work it is used for categorizing the requirements in the requirement specifications. This classification scheme is presented in Section 3.

In order to answer the third of the questions we have performed a study of relevant standards, processes and textbooks. This study is presented in Section 2.

To the best of our knowledge, very little work has been published on the state of the practice of maintainability requirements. Jan Bosch describes his experience of quality requirements, such as maintainability, by saying that they are generally specified rather weakly in industrial requirement specifications [7]. His example of a typical requirement statement is: "The maintainability of the system should be as good as possible".

Throughout this document quotes from the requirement specifications in the study have been translated from Swedish to English by us.

2. THEORY ON MAINTAINABILITY REQUIREMENTS

Several ideas on how maintainability should be handled in requirement specifications have been described in literature. In this section we present a set of such ideas. Each source has been given a subsection. In Section 5.3 we discuss how well the requirement specifications in our investigation adhere to the ideas.

2.1 General and Specific Maintainability

Bass, Clements and Kazman [5] distinguish between two different kinds of maintainability: *general* and *specific*. General is about how well software engineering principles that are known to give high maintainability has been followed. Specific is about how easy a specific set of changes can be made. Hence, general maintainability is how well prepared the system is for unanticipated changes, while specific maintainability is how well prepared the system is for a given set of anticipated changes. Bass, Clements and Kazman argue that both kinds are important.

2.2 Recommendations in IEEE 830

The standard IEEE 830-1998, IEEE Recommended Practice for Software Requirements Specifications [3], contains a suggested outline for software requirement specifications. It includes a section about maintainability:

5.3.6.4 Maintainability

This should specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices.

Our interpretation of the standard is that it states three things about maintainability requirements. First, the requirements should be placed under a separate heading, to make it clear that the purpose of the requirements is to get the desired maintainability. Second, it is attributes of the software that is to be addressed, as opposed to documentation, development process, etc. Third, the requirements should be measurable, e.g. certain modularity can be described as “module A may not communicate with module B”, and complexity can be described in terms of some complexity metric.

2.3 Recommendations in RUP

The Rational Unified Process (RUP) recommends documenting non-functional requirements in part of the requirements specification called supplementary specification. The section *Supportability* should contain requirements that will enhance the supportability or maintainability of the system being built, including coding standards, naming conventions, class libraries, maintenance access, maintenance utilities. Example: “Upgrades to the PC client portion of C-Registration shall be downloadable from the UNIX Server over the internet.” [4]

2.4 Recommendations in the OPEN Process Framework

The OPEN Process Framework [9] defines a maintainability requirement as a “developer-oriented quality requirement that specifies a required amount of maintainability, which is defined as follows: the degree to which something is able to be maintained while in use (i.e. between major releases).” They also

define a specialized type of maintainability called *extensibility*, which can occur between or at major releases.

The OPEN Process Framework recommends a clear differentiation between maintainability *goals*, which are typically inconsistent, untestable, ambiguous, etc, and maintainability *requirements*, which must be consistent, testable, unambiguous, etc. The maintainability requirements should be specified quantitatively in terms of times to locate, repair, and test minor defects or make minor enhancements, downtimes, times between maintenance actions, maintenance person-hours per specific maintenance task, maintenance hours per operating hour and maintenance cost per operating hour. This should be specified for the conditions under which the application or component will be maintained, e.g. the size and structure of the maintenance organization in terms of staff size, roles and skills levels.

Because maintainability requirements can be difficult to quantify, they may (if necessary) be occasionally specified more as desired goals than as actual validatable requirements. Nevertheless, they should not be unnecessarily specified in terms of architectural, design, and implementation constraints and the use of industry best practices that tend to produce maintainable applications and components when followed, such as layered architectures, modular software, information hiding of implementation, object-orientation and component-based development, complete and current documentation or adherence to project conventions.

2.5 Recommendations by Peters and Pedrycz

Peters and Pedrycz [15] suggest letting the software requirement specification contain target values for metrics of non-behavioral software quality requirements, such as maintainability. Maintainability is measured relative to the average values of consistency, comment count, complexity, modularity, size and degree of parallelism. These metrics are in turn defined in terms of number of conflicts with requirements, number of modules, number of comment lines, number of lines of code, number of decision points, number of variables and numbers of processors used. Peters and Pedrycz suggest formulas for calculating the maintainability value of a system based on these parameters and suggest that some constants in the formulas should be given values during the requirements phase. The requirement specification should contain the minimum value that is acceptable.

2.6 Recommendations by Jan Bosch

Bosch proposes a quality attribute-oriented software architecture method [7]. An input to this method is a requirement specification which contains quality requirements with associated scenario profiles. A scenario profile is a set of scenarios, generally with some relative importance associated with each scenario. A maintenance profile contains change scenarios, i.e. concrete requirement changes that lead to changes in the software. The scenarios can be categorized and have a relative weight that indicates its relative likelihood, i.e. the chance that the scenario occur during a time period. A fraction of an example maintenance profile is presented in Table 1.

Table 1. An example from a scenario profile [7]

Category	Scenario description	Weight
Market driven	Change measurement units from Celsius to Fahrenheit for temperature in treatment.	0.043
Hardware	Add second concentrate pump and conductivity sensor.	0.043
Safety	Add alarm for reversed flow through membrane.	0.087

Bosch's method utilizes the scenario profiles to predict values of quality attributes by doing an impact analysis of each scenario in the profile. For maintainability, the impact data of the change scenarios makes it possible to calculate the size in changed and new lines of code for the change scenarios, and using those figures the maintenance cost can be calculated.

According to Bosch's method the requirement specification should contain maintainability requirements with associated change scenario profiles. This is clearly an example of *specific* maintainability.

2.7 Discussion of Theories

We found that there is no consensus in the literature on how maintainability requirements should be handled. However, it is possible to conclude that it is a good thing to use both general and specific maintainability. While Bosch emphasizes specific maintainability and Peters and Pedrycz emphasizes general maintainability, no source argues against using both types. Since several sources argue for using both types, that could be seen as generally recommended. It is also possible to conclude that the requirements should be quantifiable, since some of the sources emphasize that this and no sources contradicts it.

There are different views on whether it is good to require the use of practices that are believed to give high maintainability, such as layered architectures and coding standards.

3. CLASSIFICATION OF MAINTAINABILITY REQUIREMENTS

Maintainability requirements can be made on several different entities related to a software system. For example, requirements can be made on the architecture of the system as well as on the source code of the system or on the process used when developing the system. In order to facilitate an understanding of the nature of requirements on maintainability we propose a classification on different entities on which maintainability requirements can be made. The idea is that each maintainability requirement belongs to one of the classes. We used the classification to categorize the maintainability requirements in our study of requirement specifications (see Section 4).

Initially we created the first 9 classes based on our common experience of software development and maintenance. The 10th class, *ownership*, was added when such requirements were found in several of the requirement specifications in the study. The classification has the following 10 classes:

1. **Documentation.** Documentation of how the system is designed and implemented can be of great value when performing maintenance of it. Requirements can be made on the documentation, for example by stating what should be documented, how it should be described (notations, languages) and technical formats for it.
2. **Architecture and design.** The chosen software architecture and the design of a system will affect the maintainability of it, and therefore it can be relevant to have requirements on them. Some architectures and design patterns are claimed to improve the maintainability, such as the abstract factory pattern [7]. Templates (i.e. patterns of software) that are repeated throughout the system can improve maintainability [5]. It is also possible to use metrics to estimate the maintainability of an architecture [7].
3. **Source code.** Requirements on complying with coding standards and having comments in the source code might have a positive impact on maintainability. Complexity metrics on the source code, such as McCabe's cyclomatic complexity [14] could be used to specify a level of acceptable complexity.
4. **Environment by third-parties.** The environment that the software is to function in is often made up by several items from third-parties, such as programming languages, databases, code libraries and operating systems. Sometimes it can be relevant to put requirements on such items for maintainability reasons, for example if the maintenance staff consists of experienced Java programmers, using Java can be a requirement for maintainability reasons.
5. **Test cases.** Having a suite of regression test cases can be a way to reduce the risk of introducing faults when performing maintenance. Therefore it can be relevant, for maintainability reasons, to require such a test suite to accompany the system.
6. **Running system.** Maintainability requirements on the running system include requirements on availability when performing maintenance, such as a limit saying how long a version change may take. For some systems it might even be reasonable to require runtime software evolution, i.e. that updates can be made without interrupting what the system is doing at all [12].
7. **Maintenance organization.** One approach to handling maintenance is to require somebody else to handle it, i.e. to have the system delivered together with some kind of maintenance services. For example, requirements can be put on how such services should be performed, on time limits for corrections, cost levels for enhancements and guarantees.
8. **Process.** Requirements can be made on how the system should be developed. For example requiring adherence to some process that claims to lower maintenance effort, such as Extreme Programming [6], or requiring specific process steps being performed such as formal inspections with a maintainability focus.
9. **Crosscutting system.** Some maintainability requirements are not tied to a specific entity, but rather

crosscut the whole system. Examples are requirements on how much work may be needed for certain kinds of changes and mean time to repair.

10. **Ownership.** In order to be allowed to perform maintenance on the system it can be important to require ownership of various items, such as source code and documentation. Generally, ownership requirements are those that make it legal for the customer to make changes in the delivered systems.

It is probably impossible to make a general claim on which entities to place maintainability requirements to get a maintainable system. For a start, it depends on the type of system. Furthermore requirements on different entities are by no means independent. For example, the Process requirement saying that Extreme Programming [6] should be used might imply that you will not need any documentation requirements and it does imply that test cases must be created. However, we argue that our classification can be of help in understanding what kinds of requirements on maintainability are common and how are they expressed. We also believe that the classification can be further developed into a process tool for authors of requirement specifications, see future work in Section 7.

4. METHOD

Section 4.1 presents the selection criteria for requirement specifications in our study. Section 4.2 describes how the requirement specifications were investigated to obtain raw data. Section 4.3 describes how the raw data was analyzed.

4.1 Selection of Requirement Specification Documents

In our study we have taken advantage of the fact that all requirement specifications used for public procurement by Swedish Government or local authorities are public documents. The authorities are also required by law to publish their requests for tenders, and all such requests are categorized depending on the type of products or services bought. The categorization is called Common Procurement Vocabulary (CPV), which is a European standard [2].

We used a commercial database [1] to find all purchases in the CPV-category “72 - Computer and related services” made by Swedish Government or local authorities from January 2003 to June 2004. From this we selected all requirement specifications that were about development of new software systems. This excludes sales of standard products (COTS). However, it is difficult to make a clear distinction of what is a modification of a standard product and what is utilization of an existing platform. We have used our best judgment to include only requirement specifications where a substantial amount of software development is necessary. A small fraction of the documents was discarded because of unavailability.

Using requirement specifications made for public procurement in Sweden for answering our research questions is a decision made primarily because of the availability of them. Commercial companies tend to have little interest in making their requirement specifications available for research. This limited scope affects the validity of the study. The validity should be reasonably good as long as the results are used for answering questions about maintainability requirements in the area of Swedish public

procurement, but the further from this area we get, the more questionable the validity gets. Although, we argue that the study should be of some interest outside this particular sector, since it tells the requirement engineering community something about a research area that has been next to unexplored.

4.2 Investigation of Requirement Specification Documents

In our study, we have tried to identify all maintainability requirements in the requirement specifications. We define a maintainability requirement as *a requirement that has as primary goal to make maintenance of the system easier*.

For each maintainability requirement, we have investigated the following:

- The entity of the requirement, according to the classification presented in Section 3.
- If the requirement is *general* or *specific*, according to the definitions in Section 2.
- If the requirement specification claims explicitly that the requirement is a maintainability requirement, e.g. by placing it in a section with the headline “Maintainability”.
- Prioritization, often in the form of “shall” or “should”.
- If the requirement is functional or non-functional.
- If the requirement is testable or not, i.e. if it will be possible to determine if the requirement is fulfilled when the system is created.

One of the major challenges when determining if a requirement is a maintainability requirement stems from the difficulty to distinguish between maintenance and usage. Using a function to put data into a table in a database is typical usage, while writing new lines of C code and thereby adding new functions is typical maintenance. But what about changing parameters in the database that alters the function of the system or using a report generator to add a new report?

As a rule we consider a requirement made to facilitate future changes (corrections or improvements) to the system a maintainability requirement. There is however an exception: Future changes that only affect “data” that the system works on are not maintenance and hence requirements that facilitate these kinds of changes are not considered maintainability requirements. We consider “data” to be anything that is not delivered with the original system. This means that two physically identical systems, which have been acquired differently, can get different classifications of the same requirement. This is reasonable since it is a result of the design of our study rather than actual properties of the system. The study is designed to answer the questions regarding the requirements used to make the maintenance easier for the system the customer bought in the first place. Example: In the requirement specification for one system there is a requirement stating that report templates should be delivered with the system. There is also a requirement stating that it should be possible to make changes in the report templates. Because the report templates were a part of the delivered system, the second requirement is considered a maintainability requirement. In the requirement specification for another system there is a requirement stating that it should be possible to create and change

report templates, but no templates are delivered with the system. This requirement is not considered a maintainability requirement since the report templates are “data” in this system.

The classes in the classification presented in Section 3 are not entirely orthogonal; in some cases a requirement could be classified into more than one class. In our selection of requirement specifications, this was rare and had little impact on the analysis of the data.

4.3 Analysis of Data

For each of the investigated requirement specifications, we present for which of the classes in our classification where there is at least one requirement.

We also present for which classes we have judged it a substantial amount of relevant requirements. We have chosen not to present a count of the requirements in a certain class, since the granularity of requirements differs a lot. Based on the data found in the study we present a qualitative analysis that answers the research questions in Section 1.

5. RESULTS AND DISCUSSION

Section 5.1 presents the requirement specification documents selected for our study. Section 5.2 presents which classes of requirements the documents contain. The qualitative analysis regarding correspondence with theories from literature is presented in Section 5.3, and a more general discussion of the results is presented in Section 5.4.

5.1 The Requirement Specification Documents

The selection process resulted in 10 systems that are listed in Table 2, where we also give each system a short name in form of a letter A-J for easy reference. To give an idea of the size of the specifications we also present the number of pages each document consists of.

5.2 Classification of Requirements

Table 3 gives an overview on which entities requirements are made for the different systems. In the table, a filled circle means that there is a substantial amount of relevant requirements in the category. An empty circle means that there are one or more requirements in the category, but we have judged that they have no, or very limited, impact on the system. There are two possible reasons for that:

- Vagueness/lack of testability. For example: “The design of the game shall be documented in a clear way” (from J)
- Very limited scope. An example: “The supplier should have competence and ability to monitor the development of standards in the health care field.” (from E). This requirement alone is too limited in scope to motivate a filled circle for the category maintenance organization.

Note that more filled circles are not necessarily better than fewer for a specific system. However, in the ideal case not having requirements in a category should be an actual choice, not something that just happened.

The classification shows us that requirements on maintenance organization are common. 8 of the requirement specifications

contain some sort of requirement concerning the possibility to buy maintenance services later in the system lifecycle.

Requirements in the categories Architecture and design as well as Crosscutting systems are also quite common, with 6 occurrences of requirements in each category. However, these are typically weaker, as only half of requirements in these categories are judged to be substantial.

Only 3 of the systems (D, H and J) have maintainability requirements on documentation, which we thought was surprisingly few.

3 of the systems (C, F and H) have requirements on the running system. C requires that updates can be handled at runtime and H requires that the system is continuously available. These are examples where runtime software evolution [12] can be a relevant technology.

3 of the systems (D, H and J) have requirements on ownership. All the three requirements concern ownership of the source code.

2 of the systems (H and I) have requirements on processes, such as ISO 9000-3 and RUP. It is however difficult to know if these requirements where there for maintainability reasons.

One issue that makes the investigation of the requirement specifications problematic is that several of them intentionally avoid specifying certain requirements, but rather ask the bidder to do it for them. For example, such a requirement can ask the bidder to specify how they intend to support addition of functionality to interfaces. Formulating requirement specifications this way is a pragmatic way to defer the decision on the requirements until the bidders have stated what they can actually offer. Nevertheless, it decreases the amount of information in the requirement specification on what is actually required.

5.3 Correspondence with literature

As noted in Section 2, there is no consensus in the literature on how maintainability requirements should be handled. In this section we relate what we found in the requirement specifications to the different theories found in literature.

All the requirement specifications contain maintainability requirements that are *general*. Only 6 of them (C, E, F, H, I, J) contain maintainability requirements that are *specific*, i.e. desirable future changes are described. An example (from H) is “Possible future extensions of the interface for output data could be possibilities to connect mobile devices for different needs”. Thus it is quite common to follow the advice from Bass, Clements and Kazman [5], that both specific and general maintainability is important. The requirements that are specific are rather informal and thus far from the structured method proposed by Bosch [7].

We found that it can be difficult to differentiate between maintainability requirements that are *specific*, i.e. that describe specific changes that might be needed in the future, from normal functional requirements. We found that such requirements tend to be a bit vague on how easy it should be to make the enhancement in question. One extreme is that it should already be implemented and ready at the push of a button, but that is probably not what is intended. An example (from I) is “Currently no discounts are used, but it shall be possible to introduce rules for discounts easily without giving rise to additional development costs”.

Table 2. An overview of the investigated systems.

	Customer	System description	Document size (pages)
A	City of Göteborg	Development and operation of PDA and server modules in real time for positioning, damage and event reporting with multimedia etc.	17
B	City of Jönköping	Billing system for drinking water, sewage and garbage collection.	6
C	Federation of County Councils	System for managing medical advice by phone on a national level. Examples of what the system includes are: redirection of calls, queue management, workflow management, medical documentation, statistics, and system administration.	147
D	Regional Council of Västra Götaland	Information network for cancer research. The goal of the system is to facilitate the creation of data repositories for clinical cancer research.	50
E	Stockholm County Council	Integration platform that is to support medical information following the patients' flow between different medical care organizations.	105
F	Swedish Customs Administration	A system for receiving and communication services for the EDI system of the Swedish Customs Administration.	37
G	Swedish Defence Materiel Administration	Web based market place for consulting services to the Swedish Defence Materiel Administration.	31
H	Swedish Maritime Administration	A ship reporting system managing obligatory reporting of hazardous goods, arrival and departure and ship generated waste in accordance with EU directives.	27
I	The City of Stockholm's Executive Office	Equipment, systems and services for handling environmental fees within the city of Stockholm. The system should support a full scale test of a fee based system for all vehicles entering the center of the city.	85
J	The Swedish Parliament	A web based pedagogical computer game that will be used to teach pupils about Swedish democracy on a national level.	5

6 of the requirement specifications (C, E, F, G, H and I) contain general requirements on architecture/design. The ideas of the Open Process Framework that maintainability requirements should not be unnecessarily specified in terms of architectural, design, and implementation constraints and the use of industry best practices that tend to produce maintainable applications when followed [9], are not applied in those requirement specifications.

None of the requirement specifications was following the standard IEEE 830 or the RUP development process. They both suggest a special heading for maintainability requirements (RUP calls it *Supportability*), which is something that only the requirement specification E has. That section only contains requirements on the internal software attributes under that heading, which is in accordance with the suggestion from IEEE 830. Requirement

specification F has a heading for "Maintenance and Support". In the other 8 requirement specifications the maintainability requirements were not gathered in a specific section. This makes it more difficult to understand if the purpose of the individual requirements is to improve the maintainability or not.

We did not find any use of target values for metrics of maintainability as proposed by Peters and Pedrycz [15].

Only one system has any functional maintainability requirements. System C has requirements on functionality for automatic program distribution and updating. That most maintainability requirements are non-functional is not surprising at all, since the nature of maintainability is typically said to be non-functional. However, it is interesting to see that exceptions exist.

Table 3. Overview on which entities requirements are made, for the systems. A filled circle means that there are a substantial amount of requirements in the category. An empty circle means that there is at least one requirement in the category, but they have limited impact due to vagueness or limited scope.

		Documentation	Architecture and design	Source code	Environment by third-parties	Test cases	Running system	Maintenance organization	Process	Crosscutting system	Ownership
A	City of Göteborg							○			
B	City of Jönköping							●		○	
C	Federation of County Councils		●				●	●			
D	Regional Council of Västra Götaland	●						●			●
E	Stockholm County Council		●	○				○		○	
F	Swedish Customs Administration		○				●	●		●	
G	Swedish Defence Materiel Administration		●					●			
H	Swedish Maritime Administration	●	○		●		●		●	○	●
I	The City of Stockholm's Executive Office		○					●	●	○	
J	The Swedish Parliament	○								●	●

As a summary, it seems that none of the ideas from literature has had any widespread impact on the state of the practice. This seem to be even more true for the more “ambitious” ideas such as Bosch [7] and Peters and Pedrycz [15]. We argue that one of the main problems of these ideas is that they imply quite fundamental changes to the requirement specifications and the requirement engineering process, reaching far beyond the area of maintainability requirements. This in turn implies high start-up costs associated with the methods.

Another probable reason for the limited impact of the literature is the lack of both consensus and available material in the area.

Many requirement engineering information sources have very little to say about maintainability requirements.

5.4 Discussion

All requirement specifications in our study contain some form of requirements on maintainability and two of them (E and F) have specific sections devoted to it. We interpret this as a widespread interest in getting maintainable systems. Around half of the maintainability requirements have priority level “shall”, from which we draw the conclusion that maintainability requirements are not thought of as less important. The other half of the requirements has the lower priority level “should”.

We judged most of the requirements as not testable. The main reason for this is that they are vague and difficult to understand. Here are a few examples:

- “It should be easy to make changes” (from B)
- “The system shall be adaptable to new health care laws and regulations” (from E)
- “The system shall be designed and implemented in a modular way (to minimize service and maintenance efforts, to minimize dependencies and to gain a better overview of the system - applies to both hardware and software)” (from I)

In the cases where testability is low due to vagueness, this is not only a problem for testing whether the requirement is fulfilled, but also a problem for the developers who have to understand what should be developed.

A total of three maintainability requirements on documentation were found in the study. Together they form an illustration of how different amount of interpretation is left to the reader. The example from J, “The design of the game shall be documented in a clear way”, does not really tell us anything. What is considered documented “in a clear way” by one person might be completely incomprehensible to another. The example from D is clearer, “The following documentation should be available at the web portal and be updated when needed: Service manual, system description, installation, maintenance description”, but it still does not say much about what the documentation should contain. H uses the strategy to rely on document contents prescribed by RUP to solve this problem: “The system should be developed and documented in accordance with RUP”. This is an example of a requirement that is testable, clear and that actually say something about what the documentation should contain.

Requirements on maintenance organization are common. This is probably a result of the fact that the organizations that are buying the systems are interested in outsourcing all work on their software systems. We suspect that this has drawn the focus away from the more technical aspects, such as test cases, source code and environment by third parties. Requirements on architecture and design were more common and it is a category that can contain very technical requirements. However, the actual requirements found in this category are typically too vague to be good technical requirements. Only two requirements on architecture and design were found to be testable. As mentioned above, requirements on technical documentation was not very strong either. We argue that it is likely that most of the authors of the surveyed specifications have missed the fact that they could benefit from the system having high maintainability on a technical level, even though they will not perform the maintenance themselves. It might be the case that they do not have the knowledge necessary to describe how such maintainability should be described and evaluated.

6. CONCLUSIONS

The main contributions of this work are:

- A better understanding of the state of the practice regarding maintainability requirements.
- A classification of maintainability requirements.

- A survey of maintainability requirement theories and recommendations.

Our investigation of 10 requirement specifications shows that it is common to have some requirements on maintainability in the domain of public procurement. Such requirements exist in all the investigated requirement specifications. We interpret this as a widespread interest in getting maintainable systems.

Our conclusion is that there is much room for improvement in how maintainability requirements are made. This conclusion is based on two things. Firstly, there is a low total utilization of the different classes of maintainability requirements. This shows that only a small fraction of the possible classes of maintainability requirements is utilized. In general, the requirement specifications would gain from using a wider spectrum of the classes. Secondly, the requirements that do exist in the specifications are generally poorly specified. Most of them are not testable, often due to vagueness. In many cases the requirements are difficult to interpret because of how they are formulated.

It is common to have requirements on *maintenance organization*, i.e. requiring the system development organization taking responsibility of maintaining the system. Technical requirements on maintainability are uncommon.

Our literature study shows that there is no clear consensus about how maintainability requirements should be expressed. Not many of the proposed ideas from literature seem to be widely used in practice. We speculate that one reason for this is that the more ambitious ideas imply fundamental changes in the requirement engineering process, which in turn implies high costs. Another probable reason for the limited impact of the literature is the lack of easily available material in the area. Many requirement engineering information sources have very little to say about maintainability requirements.

7. FUTURE WORK

We believe that the classification scheme presented in Section 3 can be further developed into a process tool for requirement engineers. One idea is to make a checklist of entities on which maintainability requirements can be made, with the intention to help the requirements engineers make sure that no important aspect of maintainability requirements is left out unintentionally. For each entity, it should also be possible to create a list of well-formulated standard requirements to serve as a source of inspiration when deciding which requirements are relevant for the system at hand. For example, some good standard requirements on the entity documentation are:

- Documentation on the database schema shall be delivered in form of entity-relationship diagrams.
- Documentation of the design shall be delivered in form of UML class diagrams for at least 50% of all classes in the system. The diagrams shall contain attributes and operations of the classes as well as relations between them.
- All documentation shall be delivered in machine-readable form as well as on paper.
- All documentation shall be in English.

Similarly it should be possible to describe standard requirements for the other entities presented in Section 3. Each standard

requirement could be accompanied with an explanation when necessary, and a description of implications a requirement might have. The requirements engineers could then select which of the standard requirements are relevant for the system at hand.

An interesting approach would be to make such a process tool available to requirement engineers and thereafter investigate how it has changed their requirement specifications.

8. ACKNOWLEDGEMENTS

This work has been supported by the Research Center on Integrational Software Engineering (RISE), which is supported by Stiftelsen för Strategisk Forskning.

9. REFERENCES

- [1] Merzell AB Sweden, www.merzell.com, 2004
- [2] "Common procurement vocabulary," European Union, 2004. <http://europa.eu.int/scadplus/leg/en/lvb/l22008.htm>
- [3] "IEEE Std. 830-1998, Recommended Practice for Software Requirements Specifications," IEEE, 1998.
- [4] "Rational Unified Process Version 2003.06.12.01," Rational Software, 2003. <http://www14.software.ibm.com/webapp/download/home.jsp>
- [5] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*: Addison-Wesley, 1998.
- [6] K. Beck, *Extreme Programming Explained: Embrace Change*: Addison-Wesley, 1999.
- [7] J. Bosch, *Design & Use of Software Architectures*: Addison Wesley, 2000.
- [8] S. M. Dekleva, "The Influence of the Information Systems Development Approach on Maintenance," *MIS Quarterly*, vol. 16, pp. 355-372, 1992.
- [9] D. Firesmith, "OPEN Process Framework (OPF)," 2004. <http://www.donald-firesmith.com>
- [10] J. Foster, "Cost Factors in Software Maintenance," in *Computer Science Department*: University of Durham, NC, 1993. <http://www.jsjf.demon.co.uk/thesis/Thesis.html>
- [11] R. L. Glass, *Facts and Fallacies of Software Engineering*: Addison-Wesley, 2002.
- [12] J. Gustavsson, "Towards Unanticipated Runtime Software Evolution," in *Department of Computer and Information Science*. Linköping: Linköpings universitet, 2003, pp. 86.
- [13] B. P. Lientz, E. B. Swanson, and G. E. Tompkins, "Characteristics of Application Software Maintenance," *Communications of the ACM*, vol. 21, pp. 466-471, 1978.
- [14] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, pp. 308-320, 1976.
- [15] J. F. Peters and W. Pedrycz, *Software Engineering, an Engineering Approach*: Wiley, 2000.