Definition and validation of requirements management measures

Thesis draft

Annabella Loconsole

PhD Thesis
Department of Computing Science
Umeå University, Sweden
Uminf ....2007
Abstract
Acknowledgement
Preface

This thesis is based on work made at Umeå University, department of Computing Science. It contains an introduction and seven papers listed below. The introduction describe the background, the research methods the contributions and a summary of the papers.

Paper 1

Paper 2

Paper 3

Paper 4

Paper 5

Paper 6
Paper 7

Paper 8

In addition to the papers included in the thesis, other papers have been produced in relation to the PhD studies (the sigce poster and the doctoral symposium paper, the SEN newsletter, the serps paper and the journal paper. During the phd studies the author has presented her work at six international conferences.

# Table of content

Abstract .......................................................................................................... i  
Preface .......................................................................................................... v  
Table of content ........................................................................................ vii  

1 Introduction ........................................................................................ 1   
1.1 Technical problem ................................................................................ 1   
1.2 Goals and hypothesis ............................................................................ 2   
1.3 Contributions ...................................................................................... 3   
1.4 Thesis outline ...................................................................................... 4   

2 Research methods ........................................................................... 5   
2.1 Software engineering .......................................................................... 5   
2.2 Methodologies in software engineering .......................................... 6   
2.3 Empirical software engineering ..................................................... 7   
2.4 Measurement theory .......................................................................... 9   

3 Requirements engineering ......................................................... 13   
3.1 Requirements management ............................................................. 14   
3.2 Requirements volatility ................................................................. 14   
3.3 Empirical studies on requirements .............................................. 17   
3.4 Empirical studies in academic environment ................................ 17   
3.5 Empirical studies in industrial environment ................................ 17   

4 Validated measures ......................................................................... 19   
4.1 Size measures .................................................................................. 20   
4.2 Requirements measures .............................................................. 21   
4.3 Overview of validated measures .................................................. 22   

5 Discussion of papers ................................................................. 43   
5.1 Paper 1 ......................................................................................... 43   
5.2 Paper 2 ......................................................................................... 44   
5.3 Paper 3 ......................................................................................... 45   
5.4 Paper 4 ......................................................................................... 45   
5.5 Paper 5 ......................................................................................... 47   
5.6 Paper 6 ......................................................................................... 48
5.7 Paper 7 .................................................................................................................49
5.8 Paper 8 replication..............................................................................................49

6 Conclusions .......................................................................................... 51
   6.1 Summary ..............................................................................................................51
   6.2 Contributions ......................................................................................................51
   6.3 Limitations, challenges and open issues..........................................................52
   6.4 Future work.........................................................................................................52

7 References .......................................................................................... 55
   7.1 References from chapter 1................................................................................55
   7.2 References from chapter 2................................................................................55
   7.3 References chapter 4..........................................................................................59

Paper 1: Measuring the requirements management key process area ............71
   1.1 Introduction ........................................................................................................72
   1.2 The Requirements management KPA of the capability maturity model...73
   1.3 The Goal/Question/Metric paradigm............................................................75
   1.4 Application of the Goal/Question/Metrics to the CMM .......................76
   1.5 Measures for the goals of the requirements management KPA ...........80
   1.6 Testing the measures in a company..............................................................80
   1.7 Concluding remarks and future directions .............................................81
   1.8 Acknowledgements ......................................................................................82
   1.9 References.......................................................................................................82

Paper 2: Non-empirical validation of requirements management measures ....85
   2.1 Introduction .....................................................................................................86
   2.2 Measures to improve the requirements management process ..........86
   2.3 Measure validation.......................................................................................87
   2.4 Conclusions and future work.......................................................................91
   2.5 References.....................................................................................................92

Paper 3: Theoretical validation and case study of requirements management measures
   3.1 Introduction ...................................................................................................94
   3.2 Software measurement...............................................................................96
   3.3 Measures validation...................................................................................97
   3.4 Theoretical validation of requirements management measures ..........99
   3.5 Case study in a SE course .......................................................................108
   3.6 Discussion and conclusion.................................................................114
   3.7 Future work...............................................................................................116
   3.8 Acknowledgement.....................................................................................116
3.9 References .........................................................................................................116

**Paper 4: Preliminary results of two academic case studies on cost estimation of changes to requirements**

4.1 Introduction ......................................................................................................128
4.2 Case studies overview ..................................................................................129
4.3 Definition of the case studies ........................................................................129
4.4 Case studies context and environment .........................................................130
4.5 Limitations of the case studies ......................................................................131
4.6 Hypotheses and plans ...................................................................................132
4.7 Results of case study one ................................................................................134
4.8 Partial results of case study two .....................................................................136
4.9 Comparison of the two case studies .............................................................137
4.10 Conclusion and future work ..........................................................................138
4.11 Acknowledgement .........................................................................................139
4.12 References .....................................................................................................139

**Paper 5: An Industrial case study on requirements volatility measures**

5.1 Introduction ......................................................................................................142
5.2 Case study description ....................................................................................144
5.3 Conclusions .....................................................................................................152
5.4 References .....................................................................................................153
5.5 Appendices .....................................................................................................156

**Paper 6: Construction and validation of prediction models for number of changes to requirements**

6.1 Introduction ......................................................................................................164
6.2 Related work ....................................................................................................165
6.3 Background ......................................................................................................166
6.4 Description of the correlational study ............................................................167
6.5 Construction of prediction models .................................................................170
6.6 Discussion and conclusions ............................................................................177
6.7 References .....................................................................................................178
6.8 Measurement rules ..........................................................................................181

**Paper 7: A correlational study on four size measures as predictors of requirements volatility**

7.1 Introduction ......................................................................................................184
7.2 Related work ....................................................................................................184
7.3 Background ......................................................................................................185
7.4 Description of the correlational study ............................................................186
7.5 Construction of prediction models .................................................................189
Introduction

In this chapter I present a summary of the thesis, describing the problem, goals and hypotheses, my solution to the problem, and my contributions.

1.1 Technical problem

Software Systems are becoming increasingly complex and large. They are often delivered too late, require higher budgets, and their quality is usually low. Therefore, building high quality software systems, within schedule and budget is a challenge for software organisations. In fact, on time and on budget projects account for one third of projects today [2]. It is well known [4] that the quality of software systems depends on early activities in the software development. One of the most important early activities is the management of requirements.

Requirements management is a continuous process performed in parallel with other requirements engineering activities. It continues through all the phases of the software development and long after delivery of the product [3]. It is the process of eliciting, documenting, organizing, and tracking changes to the system requirements and communicating this information to the project team [1] (see more in chapter 3). It is the potentially most beneficial part of the development process but it is often ignored or not done properly [5]. When requirements are not managed well, a project can fail or become more costly than intended and its quality can decrease. Requirements management practices ensure that changes can be monitored and tracked throughout the project life cycle. Without these practices high quality software is difficult to achieve. The management of customer requirements is one of the main problem areas in software development, as described in the Chaos report [7] and in an euro-
Among the requirements management practices, it is particularly important to quantify and predict requirements volatility. The reason for this is to give practitioners advance warning that a project is going off track.

One common approach that can help companies to manage their requirements is to adopt an agile software development process. Agile processes like Extreme Programming (XP) [4], SCRUM [5] and other lightweight approaches were created in response to problem domains whose requirements change. However, XP cannot be applied on a project with a large number of developers. Successful stories have been reported about projects with 2 to 30 programmers [6].

Other approaches to help large companies managing their requirements is to suggest guidelines and procedures like in the RM KPA of the CMMI [ref] or to measure the RM activities and requirements volatility (see chapter 3 and papers 5, 6, 7). Measures\(^1\) for the early development phases are the most useful since the earliest phases have the largest impact on the software development. However, few measures have been defined due to the fact that the early phases are hardly formal [4]. Furthermore, no RM measures have been validated. Measures need to be validated in order to demonstrate that they are useful. With validation we prove that the measures are related to some important quality attribute (see more about validation in chapter 2). The lack of validated measures is due to the immaturity of the measures validation field. An overview of validated measures is presented in chapter 4. Several validation methods have been presented in the literature, but there is not yet a standardised accepted way of validating measures. Some of these methods are discussed in chapter 2. Furthermore, software engineers are not sensitive to the importance of measures validation. This low sensitivity has prevented in the past from validating measures. Most of the validated measures available are late design and coding measures.

\[1\text{ In this thesis the term measure is preferred rather than metrics, as metrics connotes a generic distance measure in the mathematical and physical sciences. ieee software 97 often jeffery, there are others who say the same. See more in chapter 4.}\]

1.2 Goals and hypothesis

The general goal of this research is to provide managers and requirements engineers with validated measures that can help managing requirements and that can be used as predictors of requirements volatility. This goal can be decomposed in the following subgoals:

- to define requirements management measures
- (to theoretically validate some of these measures.)
- to show that the measures are useful to assess and predict volatility.
- to show that the measures are better predictors than subjective estimations
The practitioners (requirements engineers and project managers) would benefit from this research because the measures defined, proven to be predictors of volatility and stability, can help in understanding how much the requirements will change. In this way, the practitioners are prepared to possible changes in schedule and cost of a project and consequently they can create new plans and estimate costs and schedule of the software development. This work can help decision makers in application areas like software maintenance, configuration management, change management, project planning and tracking, ws defining baselines.

1.3 Contributions

Contribution as a whole
My contribution to requirements engineering and software measurement is the definition and validation of requirements management measures. Four (?) of the measures are proven to be predictors of requirements volatility. This is unique because, to my knowledge, no requirements management measures have ever been validated. The research methods used to obtain the results are described in chapter 2.

Contribution in details

Definition of a general set of requirements management measures
The first contribution of my research is the definition of a general and wide set of 38 software measures for the management of requirements [paper1]. The method used to define the measures was to apply the Goal Question Metrics (GQM) paradigm [] to the Requirements Management Key Process Area of the Capability Maturity Model for Software []. This set constitutes a “pick list” that can be tailored to the specific company, offering small-medium enterprises the freedom to choose a suitable subset of software measures.

Theoretical validation of ten measures
Theoretical validation of measures is still a controversial area. There is not yet a standard accepted way of validating a measure theoretically. This problem is discussed in papers 2, 3 and in chapter 2. Among the approaches to validate measures, I choose two of them and validate ten of the measures defined in paper 1 [in paper 3 which is an extension of paper 2, and maybe paper 8]. In particular I applied the “key stages of formal measurement” [] and the “theoretical validation” [].

Empirical validation
Empirical validation is performed to show that a measure is useful. Some of the measures defined in paper 1 have been used in two academic case studies. The goal of the first study was to show that cost estimations of changes to requirements based on historical data are better than intuitive cost estimations. The second academic case study was performed with the goal of showing that cost estimations of changes to requirements based on detailed impact analysis are better than intuitive cost estimations.
Furthermore, four requirements size measures have been empirically validated in four different studies (papers 5, 6, 7 and 8). In an industrial case study [paper5], we showed that there is a strong relationship between four measures of size and the number of changes to requirements. The results of paper 5 are confirmed in paper 8 where we replicated the case study above in the context of another project and compared the results with the previous ones.

The results of the empirical validation support the central hypotheses of my approach, namely that the requirements size measures are good predictors of volatility.

**Construction of reliable prediction models for requirements volatility**

Based on the previous results, two correlational studies have been performed. In the first one we construct prediction models for the number of changes to requirements while in the second for requirements volatility. In both studies, the models are proven to be reliable predictors. We also investigated the accuracy of subjective estimations of requirements volatility by practitioners. In two industrial case studies (papers 5 and 8) it is shown that the subjective estimations were not reliable.

**A wide overview of validated measures in software engineering.**

In chapter 4 a very large overview of validated measures (theoretically and/or empirically) is presented. For each study analysed, I show the measures, the attribute connected to it, the methods used to validate the measures and the environment where the validation is performed. Similar literature studies have been presented before by briand [] and by genero []. However they were limited to measures for a particular attribute.

### 1.4 Thesis outline

The rest of the thesis is organised as follows:

- Chapter 2 presents an introduction to software engineering, empirical research methods in software engineering and measurement theory.
- Chapter 3 provides basic information in the field of requirements engineering, requirements management, requirements volatility and empirical research in requirements engineering.
- Chapter 4 describe a literature review of validated measures.
- Chapter 5 presents a summary of the papers included in this thesis.
- Chapter 6 provides conclusions and future directions.

why dont we eliminate the causes of volatility? because there is always requirements volatility (explain better why)
Research methods

In this chapter I will talk about software engineering, research methods in software engineering, empirical research in SE and the problems related. Finally I will talk about the research in software measures validation (methods to validate measures and various validated measures).

2.1 Software engineering

Software engineering is defined by IEEE90 as the "application of a systematic, disciplined, quantifiable approach to development, operation and maintenance of software" [35]. We can also say that, software engineering is the application of scientific methods and theories to practical software problems. The field is young [Pour et al. 2000, 58], less than 40 years old. This is little time compared to, for instance, chemical and civil engineering which are more than 100 years old [Shaw 90, 60], not to mention natural sciences like astronomy which are thousands years old. The term “software engineering” was coined during the NATO conference held in Garmish in 1968.

Because it is a young field, software engineering is not mature yet [Pour et al. 2000, 58] and it is still debated if it should be considered engineering, craft, art, or science. Shaw wrote in the 90ties that "SE is not yet a true engineering discipline but it has the potential to become one"[Shaw 90, 60]. After fifteen years, the debate is still valid [Tomayko 2000, 67 McConnel 98, 43, Pfleeger 2005 56]. A lot of software is crafted and not all developers are software engineers. However, not all problems require an engineering degree to be solved (I do not need to be a chemical engineer to mix gin and tonic but I need to be an engineer if I want to construct a house!). According to Pfleeger, software engineering is both art and science [IEEE software 2005 56].
Among the criticism to SE, some says that it lacks important elements such as evaluation [Ticky 95, 65, Zelkowitz 98, 74], and that it is not scientific in its approaches [Fenton et al. 94, 28]. Glass et al. analysed a set of 369 papers published in six major computing journals over the years 96-01. They found that software engineering research is characterised by considerable diversity in topic, but less breadth in terms of research approach and methodology [Glass et al. 2002 31].

Suggestions on how to improve the field are given by Shaw [Shaw 90, 60] who lists five steps in order to make the field to become an engineering discipline. Other recommendations are given by Pour et al. [Pour et al. 2000, 58]. They describe nine key infrastructure components which can help software engineering to become a mature profession. They also suggest a deeper partnership among industry, academy and professional societies. The need of academy to collaborate with industries is also recommended by Basili [Basili 96, 10] and by Shaw [Shaw 90, 60]. She encourages the coupling between science and commercial practice. According to Basili, software engineering researchers should use industry based laboratories in order to observe, build, and analyse models. The need is reciprocal, industries need to build quality systems, therefore researchers and practitioners should collaborate closely to help each other. However, the two communities are often out of contact one another [Zelkovits et al. 98 sew proceedings, 75]. Since industries are different from each other they can provide only local models. Therefore researchers need many experimental laboratories in different context in order to produce general results which can be applied in many contexts [basili 96, 10].

2.2 Methodologies in software engineering

Software Engineering is a cross disciplinary field, we can find in it elements of psychology, social science, management, economics and others. Therefore research methodologies have been adopted from other fields. The consensus on SE research methodology is not reached yet in fact there is a vast variety of methodologies available in the literature. One part of the problem lies in the difficulty to define the boundaries of the field [Natt och Dag 2005, 48].

A widely used classification of research methodologies is the following:

- **the scientific method**: the world is observed and a model is built based on the observation, for example a simulation model.
- **the engineering method**: the current solutions are studied and changes are proposed and then evaluated.
- **the empirical method**: a model is proposed and evaluated through empirical studies.
- **the analytical method**: a formal theory is proposed and then compared with empirical observations [Wohlin et al. 2000 page 4 70 which refer to glass 94], [basili 93], [Zelkovits 98 74 which refer to adrian 93].

The engineering and empirical methods can be seen as variation of the **scientific method** [basili 93 da Wohlin 2000 pag 49 70]. In fact, Basili [basili 96, 10] talks only of the analytical and experimental methods (which correspond to the scientific method) (paradigm and meth-
od seems to be the same things). Within the experimental method he describes the evolutionary and revolutionary approaches. The first is based on improving an existing method, while the second is based on inventing a new method.

Basili et al. describe the scientific method as an iterative process of model building, prediction, observation and analysis. The method requires that theory can be trusted only after numerous attempts to falsify it. Unfortunately in computer science and in software engineering there is no balance between model construction and evaluation i.e. there are more models constructed than models evaluated [basili, 14] (I think also zelkovits 98 says the same).

If I want to apply these methods to my research: One (escom) partial engineering method. Two (workshop): partial engineering? Three (Uminf) partial engineering and empirical method?, Four (smef) empirical, Five (apsec) and Six (joist): scientific method.

2.3 Empirical software engineering

Software engineers have often to take important decisions. For instance the choice of the best technique to find faults, the best tool to support configuration management, or the choice of the best requirements specification language, or others. These decisions should be answered in an objective and scientific way [fenton pfleeger 29]. One way is to ask an expert or experienced person but this is often not based on scientific research (fenton pfleeger glass? 28). The best option is to perform an empirical study.

An empirical study is an activity which involve collecting data about some aspects of software development, analysing the data, and drawing some conclusions from the analysis. This is done for the purpose of discovering something unknown, or testing an hypothesis, or constructing and validating quality models [basili lanubile shull 14].

In general, empirical software engineering is needed to make evaluations of results. Since it is based on observations, empirical research is closer to the real world compared to theoretical or analytical research (Harrison et al. 99 33). The importance of experimentation in software engineering is also stressed by Tichy (tichy ieee computer 98, 66) and Pfleeger (ieee computer Oct. 99, 54) who says that no science can advance without good experimentation and measurements. Furthermore, in two workshops on software engineering research strategies organised by the national science foundation, big importance was given to empirical research to validate the theoretical principles where many excellent ideas are originated but not used [basili 2000 16].

Unfortunately, the amount of empirical research is little compared to other fields and the way it is conducted is poor (Jørgensen and Sjøberg 2004 [37], Tichy 95, [65], zelkovits 97?, Briand et al. 99 [17], and many others...). Tichy (Tichy 95), [65] and Zelkovits (Wallace e zelkovits 97), [69], have been very influential and have condemned software engineers for not validating their results. Tichy et al. analysed 400 research papers in computer science. They found that among the papers needing an experimental validation, 40% did not have any at all. In particular for software engineering, 50% of papers required validation but did not have any. (in short: The low amount of validated results seem to be a serious lack in computer science and in particular in software engineering) [Tichy 95 65]. Wallace and zelkovits analysed
612 papers, 20% did not have empirical validation and one third had a weak form of validation (compared to 5-10% in other fields) [Wallace and Zelkovich 97, Wallace and Zelkovich 98, 69, 74]. Glass et al. have found that very few evaluations and very few case and field studies have been done during the years 96-01. Often computer scientist tend to adhere to an ad-hoc evaluation of their research [Harrison et al. 99, Fenton, Pfleeger, Glass 94, 28] instead of evaluating it through an empirical study. The reasons can be many, for instance, it is hard to control the large amount of variables in empirical studies (Harrison et al. 99, 33). Basili et al. state that empirical software engineering is difficult, complex and time consuming. Software development is a human based activity therefore any variation of the subjects in empirical studies can affect the results of the study (any variation in human ability tend to obscure the experimental effects) (Basili, Lanubile 14). The software built is always different from the previous one. Therefore we are often unable to collect a high number of data points that permits sufficient statistical power to be able to accept or reject hypotheses (Basili, 96 10). Drawing general conclusions from empirical studies is difficult because the results are dependent on many context variables which make difficult to apply the study in another context (goals, context, process are usually different). Many studies are isolated and it is hard to understand how widely applicable are the results of these isolated studies. Therefore replicating empirical studies is important. However, even when replications are run it is sometimes hard to understand the commonalities and the differences so that we can draw general conclusions. Another difficult aspect of empirical software engineering is to find subjects participating to the studies. The people with knowledge in software engineering is a very small percentage of the human population and only a small amount of them is available for experimentation (Basili, Lanubile). Furthermore, we usually need experienced software engineers. But how can we evaluate the expertise on software engineers? One way is to make qualitative studies and interview the subjects about their knowledge. All these factors make the number of subjects be small and often we see studies with an amount of subjects that does not allow to achieve an high statistical power. For these reasons, experimenters often choose students as subjects (from TSE 25(4) 14). The reliability of using students as subjects is still debated [ref1]. Höst et al. [Höst et al. 2000, 34] have performed an experiment to evaluate the differences between students and professional subjects. They found that there were no significant differences between the groups. However, there are only few studies of this kind. In general the reliability of interviewing subjects need to be checked with methods like Cronbach’s alpha [ref2], Cohen kappa [ref3], the Delphi method (Dalkey et al. 1963, 25) etc.

In order to improve the empirical research in software engineering, many guidelines and suggestions are available in the literature. An introduction to empirical software engineering can be found in (Juristo 36, Wohlin et al. 70, (Kitch Sigsoft 40)), and general guidelines are described by [Perry 2000, 52], and [Kitchenham et al. 2002, 38]. More specifically, guidelines are available on how to perform experiment in computer science (Tichy 95 65), in software engineering [Basili 86, 11], empirical studies in process improvement [Ott et al. 99, 51], how to collect valid data [Basili Weiss 84, 12], and replications of empirical studies [Basili Lanubile ad Shull 14].
Furthermore, Harrison (harrison et al. 99 33) propose to create a database of empirical research projects, so that it is easy to classify the studies and compare the results. [Briand et al. 99, 17], propose (?) a methodology of conducting empirical studies in the area of object oriented software development and maintenance. Basili [isce 96 10] suggests to work with the Quality Improvement Paradigm (QIP) (used by SEL at NASA) which combines the evolutionary and revolutionary approaches. An historical perspective of empirical studies in software engineering can be found in zendler 2001 [76].

Empirical studies can be quantitative or qualitative. Quantitative are those used to gather data like numbers or information in some finite scale, while qualitative studies are those that collect information like text and pictures. Empirical studies can also be classified as descriptive, correlational, cause-effect. The kind of subjects used (novice, experts), and where they are done (in vivo, in vitro) [zelkovits wallace 98, 74]), the amount of control (observational, or controlled experiment) [basili icse 96 10]. A good detailed taxonomy of empirical studies is done by zelkovits and wallace (ieee comp 98, 74). The most used classification is surveys, case studies, experiments.

### 2.4 Measurement theory

An important role in empirical software engineering is played by software measurement. Software measurement allows for defining quantitatively the degree of success or failure, for a product, a process, or a person. It facilitates the identification and quantification of improvement, and the lack of improvement or degradation in our products, processes and people. It helps to make meaningful and useful managerial and technical decisions, identify trends, and make quantified and meaningful estimates. Even when a project runs without problems, measurement is necessary because it allows to quantify the health of the project (Fenton and Pfleeger, 1996, 29). More advantages and disadvantage of software measurement are described in [loconsole internal report]

When we decide to measure our projects or to perform an empirical study, we have to define software measures and a model associated with them. The model has to describe the entity and attributes being measured, the domain and range of the measures, and the relationship among the measures. We also need to know whether the model built is made for assessing or predicting process characteristics [pfleeger 97, 53]. In order to construct this model we need frameworks to define measures and procedures to validate the measures defined.

**Measures definition**

The most common method used in software engineering to define measures is the Goal Question Metrics (GQM) (basili 13, maybe van solinghen 68?). The GQM is a measurement framework which assists software engineers to define measures and to interpret the data. Other measurement frameworks have been defined in the past. Lott (42) describes seven different approaches to define measures, all of them are goal oriented based, but among the seven, only
the GQM has become a “widely used and well respected approach” (page 740 futrell shafer, 30) (harrison et al. 99, 33). The GQM includes a template for the goals definition that can be used to articulate the goals of any study (basili lanubile 14).

Extensions of the GQM are available in the literature. The V-GQM (Olofsson 2001, 50) is a method for analysing a GQM study after the data has been collected, in order to validate the study. The V-GQM is a support for running several subsequent studies. The M3P is an extension of QIP/GQM paradigm [ref?], which includes additional features designed to support known success factors of measurement programs and to support data measurement, analysis and interpretation [offen jeffery, 49]. The GQM/MEDEA (briand basili, 20) is an approach to define measures with the GQM and validate them theoretically and empirically. However, all these extensions are not as popular as the original GQM.

**Measures validation**
Measurement theory addresses the issue of whether the measures we are using are valid with respect to the attributes they are purport to measure.

Many measures have been defined and used without being validated. Measures need to be validated because they are not useful if their practical utility is not proven empirically [1, 4, 9]. The lack of validation of measures has led to a lack of confidence in software measurement and this is one of the reasons of the poor industrial acceptance of software measurement [bieman et al., 15]. There are two different kinds of validation which should be performed for a measure to be valid, they are generally called theoretical and empirical validation. In empirical studies, these two kinds of validations are called construct validity and predictive validity. The validation should be done for both the dependent and independent variables. However, there is not yet a standard way to validate measures.

**Theoretical validation.** There are 2 main approaches to theoretical validation [morasca 2001, 46]:

- the representational theory of measurement and
- the property based approaches.

The representational theory of measurement is based on the idea that the properties of the attributes in the real world should be maintained by the measures in the mathematical world [pfleeger jeffery... 53]. This imply the definition of an empirical and numerical worlds and the construction of a mapping between the two worlds. This kind of validation is also called internal validation [baker et al. 90, 8] [fenton and pfleeger 29] [zuse 78], validation of measures for assessment [fenton 27] [bieman 15] and theoretical validation [briand ese 19] [briand et al. 95, 18].

The property based approaches are usually founded on a number of axioms the measures must satisfy and on properties of measurement scales [Shneidewind, 2002, 62]. This kind of theoretical validation is also called axiomatic [poels and dedene 57], analytical [melton et al. 90, 45], and algebraic (shepperd and ince) validation. Among the most popular set of axioms, Poels and dedene [57] consider a metric as a distance function and define a set of axioms to
validate a measure. Weyukers [69] identified nine properties which are useful to evaluate syntactic software complexity measures. Her properties have raised an intense discussion [briand ese 19], [briand ieee 21], [zuse ieee 77], [kitch et al. 39], [kitch replay to...41] [zuse book 78], [morasca et al. 47] [gursaran 2001, 32] [sharma, 59] [zhang, 72] [hitz and montazeri], [cherniavsky and smith], [zhang and xie]. Schneidewind [63] describes a methodology consisting of six mathematically defined criteria. The methodology integrates (the validation of ??) quality factors, metrics and functions.

Two popular validation approaches combine the property based with the representational theory of measurement. Briand et al. [ieee 21] describe properties of measures of size, complexity, length, coupling, and cohesion. They state that the representation condition is an obvious prerequisite to measures validation. Their work has been criticized by Xia [99, 71], because some of the concepts defined are still vague. A broader approach to theoretical validation is taken by Kitch et al. They suggest to validate attributes, units, instruments, and protocols (the data collection procedures) [kitch et al. 39]. One of the properties suggested for the attributes validation is the representation condition.

I can conclude saying that there is no agreement on a standard definition of theoretical validation of a measure [me and pfleeger jeffery 53]. I believe that different attributes have to be validated in different ways, but the representation condition must be satisfied always ((the approaches of briand ieee 21 and kitch 39).

**Empirical validation.** Software measures are empirically valid if there is a consistent relationship between the measure and an external attribute [zuse 78]. This is usually demonstrated by performing an empirical study. The empirical validation can be a proof of a correlation between the attribute and the measures. In this case, statistical methods like Pearson, Spearman, Anova, Kendall, and others are applied. Alternatively we could construct a prediction model by applying different kinds of regression methods (linear, logistic, mars, trees,).

According to Briand et al. [briand 95, 18], empirical validation implies data collection, identification of measurement level (scales), and choice of analysis method to formalise the relationship between the internal and external attributes. The most detailed methods of how to validate a measure empirically are described by El emam [el emam, 26], Briand and wust [22], briand, wust daly etc [23].

The empirical validation of software measure is also called **external validation** [fenton pfleeger 29], [zuse 78], and **predictive validation** which is based on checking the accuracy of the prediction model [baker biem an 8, bieman 15] [fenton, 27]. According to kitch et al., basic statistical techniques such as correlation analysis can be used to investigate relationships between attributes [kitch, 39] but we have to be aware if we are validating an association between attributes or a causal relationship.
When performing empirical validation however, the connection between internal and the external attribute values are seldom sufficiently proven. As stated in [Fenton Pflieger, 29], there are two reasons for this; 1) it is difficult to perform controlled experiments and confirm relationships between attributes and 2) there is still little understanding of measurement validation and the proper ways to demonstrate these relationships.

The procedure to follow for experimental validation varies significantly depending on the purpose of the measures (evaluation or prediction), the type, and the amount of data collected [Briand et al. theoretical empirical., 18].
Requirements engineering

Very often, complex software systems are delivered late, over budget, and do not meet the needs of the stakeholders. These failures are often due to problems with the requirements for the system [kotonia sommerville]. Project requirements are the biggest cause of trouble for software projects. Many studies demonstrate that when there is a failure, the requirements are usually their cause [from bray page 7, but from glass]. Also the studies reported in the caos report say the same.

Requirements are sentences which describe a functionality (functional requirements) or a property (non functional requirements) of a system. Both kinds are defined during the early stages of a software development process. Some researchers suggest requirements to describe what the system should do rather than how. This is an appealing idea but not realistic in practice. It happens often that the system to be developed must be compatible with the environment and the organisational standards. In these cases we need to specify policies and other constraints on the system [kotonia sommerville].

There are several views of requirements engineering. The purpose of requirements engineering is to convert a poorly defined problem into a well defined problem. Davis defines it as "all the activities up to but not including the decomposition of the software into its architectural components". This is true but it does not say much about what activities it includes: requirements engineering has the task of investigating and describing the problem domain and requirements and designing and documenting the characteristic for a solution system that will meet those requirements [bray book]. Requirements engineering is the systematic process of eliciting, understanding, analysing and documenting these requirements [kotonia sommerville].

In this chapter we will describe some of the activities performed in requirements engineering focusing on those related to our research (boh, maybe I should write it better).
3.1 Requirements management

Requirements management is the science and art of gathering and managing user, business, technical, and functional requirements within a product development project. The project could be for a new consumer product, a web site, or a software application. In all these cases, the four classes of requirements should be represented. If they are not, the project will suffer user or consumer rejection to some degree. (From here, we’ll say ‘user’ for simplicity. But the rules apply even if the tool is a new type of hammer rather than a software application.) from wikipedia

Rq management is the process of managing changes to the system requirements. Requirements for a system always change to reflect the changing needs of system stakeholders, changes in the environment where the system will be installed, changes in laws and regulations etc. These changes have to be managed to be ensure that they contribute to the business needs of the customer.[kotonia soomerville]

The main requirements management activities are change control and change impact assessment. Change control is concerned with establishing and executing a formal procedure for collecting verifying and assessing changes. Change impact assessment is concerned with assessing how proposed changes affect the system as a whole.[kotonia soomerville]

but there is still a lack of measurement for the management of evolving requirements (according to lam and shankararaman).

in each section I should say why I am talking about this.

3.2 Requirements volatility

In the Oxford Dictionary, the term “volatile” is defined as “easily changing” [15]. The concept of software requirements volatility is not yet well defined. In [33] it is defined as the “intensity and distribution of changes” (which is the ones followed in this paper, see also section 6.2). One definition states that requirements volatility is the ratio of requirements change (addition, deletion, and modification) to the total number of requirements for a given period of time [36].

Research in requirements volatility area can be divided in two main streams: study of the impact of volatility on the project or development process, and construction of models to predict volatility. Our study belongs to the prediction models group. To our knowledge, none of the prediction models investigate whether the actual volatility corresponds to the perceived volatility, which is our contribution.

Related work
We describe here some of the research done in volatility with a focus on the measures used (see table 1 for a brief summary).

Costello and Liu [7] describe the role of measures in an integrated approach to system and software.

Javed et al. [18] investigate the impact of pre/post release changes on the defect density. Their findings indicate that there is a significant relationship between pre/post client release changes and software defects.

Malaiya, and Denton [24] analyse the influence of requirements changes in time by examining the consequences of additions, deletions, and modifications to software. The results show a greater impact of changes on defect density if the changes arrive late in the process.

Most of the approaches are based on the evaluation of software requirements specifications and therefore are based on historical perspective. Bush and Finkelstein instead have described a pro-active approach [5]. Their approach to predict the volatility of the requirements is based on the possible evolution of the requirements. They start from an initial set of requirements and create worlds of possible evolutions of the initial set of requirements rather than analysing historical projects. They report results from an industrial case study validating their approach. Furthermore, they describe in [6] an approach to requirements analysis that would provide predictions of requirements stability.

Ferreira et al. [9] present a simulation model developed to help project managers understand the effect of requirements volatility. The results show that the simulator can be used to demonstrate the effects of requirements volatility on a software development project.

Henry and Henry [14] describe analysis techniques to assess and predict process and product characteristics. The results provide valuable information for predicting process and product characteristics.

Harker, Eason, and Dobson [12] classify requirements as stable and changing and propose a further classifications of the origin of changing requirements. This classification helps to identify different approaches to handle changing requirements. Their conclusions is that the approach to be used to changing requirements depends highly on the nature and source of change.

Nidumolu [26] define three levels of uncertainty requirements: instability, uncertainty, and analysability. In the study presented he found out that while requirements uncertainty increases the performance risks, software development standard reduced these risks.

Lam et al. [20] main contribution is to describe a framework to manage change called Change Maturity Model. They show a general taxonomy of change not only for requirements. They define the metric “change volatility” which is a measure of the maturity and stability of system. The only requirements measure is the requirements change density defined as the number of changes (additions deletion and updates) to a requirement.

Pfahl et al. [29] present a simulation model to demonstrate the impact of unstable software requirements on project performance and to analyse the cost of stabilising requirements. Based on the simulation model they show that requirements volatility is extremely effort consuming.
Henderson-Sellers et al. [13] propose a framework for developing use case metrics. They list the possible attributes of requirements, and measures of size for use cases (number of atomic actions in the main flow, in each alternative flow, the longest path between the first and last atomic action of the use case). These measures can be useful to measure complexity and effort.

All the measures proposed here are very useful when the requirements are well documented. However, even when the requirements are well documented, the measures have to be tailored to the particular organisation because each company has its own style or way of documenting requirements. The measures can only give suggestions of what can possibly be collected in a company and the reasons to collect them.

Approaches to manage volatility

Volatility of requirements depends on many parameters for example the size of the company, the kind of process, the product we are building, the phase of the life cycle, the volatility of the market, the technology, etc. In general requirements volatility depends on the methodologies, type of product, and processes used. The reason to estimate the volatility of the requirements is usually to predict the risk that the project is going off track. One approach to use volatility measures is to set thresholds for how much requirements volatility is reasonable throughout the life of the project. For example at the beginning of a project when we are in the phase of discussing requirements, we should expect volatility to be high for a healthy process. Low volatility might mean that the requirements engineering process is stalled. Projects with low volatility are typically students projects, where the requirements are well defined and stable. In real industrial projects, when we begin to design and implement we hope for stability at least for the parts of the projects we are designing or implementing. If the data collected exceed the threshold, the measure is suggesting us to analyse deeply the project, understand it, and eventually take actions to adjust the process. Two factors contributing to requirements volatility is process stability and maturity. If the processes are not defined and stable and the company is weak at requirements engineering activities, the company will have to face high percentages of volatility and the project will be unpredictable. The threshold in this case will be higher than those of a mature company. Furthermore, if the project is small, volatility is more easily handled therefore the threshold levels should be put higher.

Another approach to measure volatility is to check trends. The numbers could be captured on fixed project intervals (daily or weekly) and the rate of change plotted over time overlaid across the project phases.

A further approach is to report the percentage of changes in time and life cycle phases by using a chart and to report the number requirements changes (number addition, deletion, modifications) by month or phases. Finally, to investigate why the changes occur and how critical are the changes. In general it is suggested to check qualitative aspects.

For Lauesen [] each requirement should have a priority assigned and the expected frequency of change which can help to identify the functions that should be easy to modify...
3.3 Empirical studies on requirements

in each section I should say why I am talking about this.

I do not want to talk in details of empirical studies on requirements, it would take a lot of
time. The best I can do is to classify the empirical studies in some way (for instance by re-
search area?)

Zowghi and Nurmuliani [42] perform an empirical study on requirements volatility and
its impact on the project’s performance. Their study presents an aspect similar to the ones pre-
semed in this paper, that is they measure the perceived volatility by the developers in different
phases of the software development. Their results show that frequent communications be-
tween the users and the developers have impact on the stability of the requirements.

Nurmuliani et al. [27] propose a detailed taxonomy of changes and present a case study
to identify the causes of requirements volatility (changes in customer needs, developers’ in-
creased understanding, and changes in the organisational policy).

A deep study on effects of requirements volatility on the project planning process and re-
lated measures has been done by Stark et al. [36, 37]. They define volatility as the rate of add-
ed, deleted, and changed requirements per total number of requirements. Furthermore, in
[37] they collect data on 40 software releases to understand the source, the magnitude and
the effects of changing requirements. Among their findings, additions are the most common
requirements change; customers and suppliers change requirements equally; an accurate
measurement program is useful for preventing and controlling requirements volatility.

Ambriola and Gervasi [1] propose two measures, one of them is requirements stability
based on the amount of information contained in requirements at a certain time. They de-
scribe the validation of the measures, showing that the stability measure had a high predictive
value.

3.4 Empirical studies in academic environment

3.5 Empirical studies in industrial environment
Validated measures

A software measure is a quantification of some property of a software artifact. Software measures are useful to control, manage, and predict the state of the software and of the process. Usually software measures are used for assessment and prediction. In the first case, we evaluate the state of an entity in order to know its quality level and to understand what has happened in the past. In the second case, we would like to predict an attribute of some entity, for instance the cost of a development project. Measurement for prediction requires the construction of a mathematical model [Fenton Pfleeger].

In this thesis I use the term “measure”, which is more general compared to “metric”. In measurement theory, a metric is associated with the distance between two points.

Software measures can be associated with different kind of entities i.e. product, process, and resources. Measures can be direct or indirect [29 Fenton Pfleeger and 78 Zuse]; a measure is direct if data can be collected only by observing the entity, for instance “number of requirements”. Indirect measures are composite, for instance fault density is equal to number of fault divided by the Lines Of Code (LOC). Measures can also be classified as subjective and objective. Ideal software measures should be simple, and objective, but the most important attribute is its usefulness. A measure is shown to be useful by demonstrating that it is connected to some internal structural properties of an entity (theoretical validation). It is also possible to demonstrate that the measure is connected to some external quality attribute (empirical validation).

The procedures to theoretically and empirically validate a measure have been described in section [2.4]. In this chapter I will classify some validated measures from the literature based on the techniques used (theoretical, empirical), and based on the environment where the validation was performed (academic or industrial).
4.1 Size measures

One of the most common internal product attributes is the size of software. It is usually measured by counting something: lines of code, number of methods, of objects, of files, of specification pages etc. Size is measured for several purposes, for instance to predict productivity, effort, cost, number of defects, number of changes etc. Measures of size can also be used to normalise other measures (e.g. defect density, change density, number of defect per thousands LOC, and similar). Size of software artifacts is measured during any phase of the software development. However, most of the research has been done to measure the size of source code. The traditional way of measuring is by counting the lines of code. LOC is a measure of length and it is correlated to effort (hettel 93 see zuse page 412). It is a simple measure, easy for managers and developers to understand and to use. However, we have to be aware of counting LOC. It is a late measure, before the source code of the system is written there are no LOC available. Usually, it is in the beginning that we want to predict the size of the software. LOC is language dependent, i.e. different languages can lead to different numbers for the same algorithm. Furthermore, lines can have different complexity and therefore require different effort. Some programmers use blanks or comments in order to improve the understandability and readability. Lines of comments require less effort to write compared to lines of code. In general we should set rules counting blank lines, comments, data declarations etc. The way we define the rules depends on how we plan to use the measure. If we need to know the storage space necessary for a program, we need to count the source instructions and the comments. For other purposes, like estimating cost, we might need to count only the source instructions.

To overcome some of the disadvantages of LOC, there are measures of functionality like Function points (FP) [albrecht], and similar approaches (feature points ref?, object points ref?). The FP method is based on measuring the functionality of specification documents, therefore it is an early measure. FP are independent from the programming language used. Function points have also drawbacks like for instance subjectivity and double counting (the internal complexity maybe accounted twice) [fenton pfleeger]. Nevertheless, the FP method is successful [zuse] and provides good estimates especially for business information systems [moser 99].

For object oriented languages, counts of classes and methods are usually more accurate productivity estimates than LOC (see fenton page 254, pfleeger 89). These counts can be done early in the software development process i.e. in the specification and design phases. Specifications and design size measures are the most useful because they allow for early pre-

---

2. Fenton and Pfleeger suggest to describe size with 3 attributes: length (physical size), functionality (what the user actually gets) and complexity [ref].
3. The calculation of FP is made by summing the number of input, output, inquires, internal and external files, each multiplied by a weight. The number obtained is multiplied by the sum of fourteen adjustment factors. These factors (like for instance reliability and performance) are rated subjectively on a scale 0-5 from the developers.
4. See [fenton pfleeger] and [zuse] for a detailed list of drawbacks of the FP method.
pected measures of cost, effort, and final size. What is measured in these phases is mainly the docu-
mentation produced. Specification and design documents are usually made by a combination
of text and diagrams. Therefore, when measuring the size in these phases, we need to use dif-
f erent measures. For instance the length in specifications and design documents can be meas-
ured by counting pages of specifications or number of requirements. Among the most known
object oriented design size measures are number of attributes, methods, children, and func-
tion points for object oriented (3D function points [whitmire], predictive object points
[minkiewicz], object oriented function points [antoniol et al.], and class points [costagliola et
al.]).

4.2 Requirements measures

Little literature is available on requirements measures.

The most common attributes

The most common measures/the most famous set of requirements measures (Costello and
Liu)

During all the phases of software development we can measure the product (the require-
ments and design documents, the source code, the test cases etc.) and the process. Require-
ments product measures are usually used to evaluate the quality of the requirements
specifications documents. Davis et al. 93, explored deeply the concept of quality of a software
requirements specification (SRS) and present 24 attributes and the corresponding measures
for SRS.

Process measures are usually defined to evaluate the quality of the process. For instance we
can measure the “kind of documentation” or the time spent in a certain phase to see how
much time is the process consuming. However, the distinction between process and product
measures is not so sharp. For example in case we measure volatility......

Requirements management measures raynus, baumert mc winney, my paper

Functional/non functional requirements measures?

Requirements size

The size of a requirement document can be computed at varying levels of granularity, because
the requirements documents are usually organised hierarchically.

Counting the number of Requirements is widely used to measure the size of a software
application [morasca handbook seke]. The variation in requirements count over time has also
been used to quantify the volatility of requirements which is a frequent phenomenon in in-
dustrial software development. [morasca seke handbook], rosenberg.

Use case measures: Use case points are a successful method for estimating software devel-
opment effort [bente anda 2002 e icse 2005, edward r carroll 2005].

In my studies [apsec, joiast] I used requirements size measures (number of lines and
number of words in use case models) which are very similar to LOC. Hence my measures have
some of the advantages and disadvantages of LOC. In particular, depends on the language
used and formatting style. As pointed out by Armour [2], what I actually want to measure is
how much knowledge there is in the system or file. Unfortunately, there is not yet an empirical way to measure knowledge. Different lines can have different complexity in use case models. A possible measure could be use case points (UCPs) [34]. However, UCPs are not generally applicable. The definition of UCPs is based on a classification of use cases and a number of environmental factors (similar to the adjustment factors in FP). This information was not available for the projects analysed. The classification of use cases is a subjective activity, therefore it is complex and hard to collect UCPs automatically.

4.3 Overview of validated measures

Hundreds of measures have been defined but few of them are validated. The reasons for this is the young age of software measurement field, the lack of clear definitions of validation and the lack of agreement among the researchers.

In this section I present a survey of validated measures. Piattini et al. [192] have performed a survey describing five sets of measures for conceptual data models. Another more extensive survey has been performed by Briand and Wust [22]. They focused on the empirical validation of object oriented measures as predictors of some quality attributes and evaluation of the prediction models. Furthermore they describe a procedure to construct prediction models.

My survey has a wider scope compared to the ones of Briand and Wust, and Piattini et al. [22, 192], whose scope was limited respectively to object oriented design and conceptual data models. My literature search was conducted mainly through digital libraries (ieee and acm dl and google scholar restricted to “Engineering, Computer Science, and Mathematics”). The keywords used in the search were “validated measures”, “empirical validation”, and “theoretical validation”. I examined 55 studies published between 1995 and 2006. The main journal and conferences contributing to this survey are IEEE TSE, and the conferences Metrics and ICSE. I searched also in the “inspec” database, with keywords: empirical validation (1172 results) restricted to: "software quality" (44 results), software metrics (94 results), statistical analysis (66 results), regression analysis (27 results). The same for theoretical validation..... After that, selected the titles related to software and read the abstract of the papers selected (xx in total)

Table 1 shows the most important characteristics of the validated measures found in my literature search. Each row in the table represents one conference paper or journal article and describes one or more studies. For each study or collection of studies I show:

- the source of the study (column Reference);
- the validated measures (i.e. the set of independent variables) (column Measures);
- the external attributes related to the measures (i.e. the dependent variables) (column Attribute);
- the entity (software object) under analysis (column Entity);
- the theoretical validation. In this field I describe whether the measures are theoretical validated and the validation procedure used. I show also the source, in case the measures were theoretically validated in previous studies (column Theoretical);
• the empirical validation. In this field I describe whether the measures are empirically validated and the kind of analysis applied (univariate, multivariate or other) (column **Empirical**);

• the environment. Here I write if the study was performed in an academic or industrial environment (or if the data was collected from an academic or industrial developed system) (column **Environment**).

By analyzing the table, we can observe that the most explored **measures** in the literature are the Chidamber and Kemerer (CK) set [ref]. About half of the studies are validation of object oriented measures (coupling, cohesion, and inheritance). Circa 10% of the studies explore data base measures, most of them have been performed by the Alarcos group in Spain. The remaining studies (40%) seem to be isolated, i.e. the entities are rarely investigated and the studies are not replicated. In particular, there are no studies on requirements or requirements process except those of briand [109] and loconsole [internal report, apsec, joiast] while it is in the beginning that we want to predict future situations.

A widely investigated **dependent variable** is maintainability (45%), followed by reliability (30%, measured often as fault proneness and defect density), and effort 25%. Very few studies investigate attributes like project performance, similarity, volatility, and usability.

The most common approach to **theoretically validate** measures is the axiomatic validation defined in [21] and the ones in [57]. Among the studies reporting a theoretical validation (29 studies), about 70% describe some kind of axiomatic validation while the measurement theory approach is used in 30% of the studies. However, 3 studies (circa 10%) report both kinds of theoretical validations (therefore, the sum of the percentages become higher than 100%).

On top of the 29 studies, other six discuss the construct (or theoretical) validity of the measures without using any validation process.

Among the **empirical validation** (49 studies), more than one third (35%) show univariate analysis while 36% show some kind of multivariate analysis. Other kind of analysis is shown in 15% of the studies (for instance comparison of descriptive statistics, expert opinion, or others). Comparing the theoretical and empirical validation, 6 studies (11%) report only the theoretical validation, while 25 have only the empirical validation (45%) and 24 have both the validations (44%). In total, 90% of the studies have at least the empirical validation, while 45% have at least the theoretical one. About half of the empirical studies are performed in an academic environment or the software system from which the data was obtained, was developed in academic environment.

**Difficulties in performing the literature review.** I found it difficult to limit the search to specific libraries or search engines. There are studies (like [apsec, joiast]) where the measures were theoretically validated in a previous study that did not appear in the chosen digital libraries. I decided to include also those studies (not appearing in the dl) in my literature review.

The distinction between theoretical or empirical validation of software measures and other kinds of validations is not so sharp. The keywords chosen for the search in the DLs, are not always mentioned, even if the studies report investigation of relationship between measures
and attributes. Regarding the theoretical validation, some researchers use their own procedures to theoretically validate measures. I have disregarded those studies because the procedures used are not widely recognised.

Furthermore, sometimes the same studies are described singularly and as a family of studies; sometimes one study investigates measures collected from different previous studies. 84 studies is now.

**TABLE 1:**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretical</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>---Abrahaо et al. [79]</td>
<td>size and structural complexity of web applications 8 metrics for navigational maps, 5 metrics for navigational context</td>
<td>Maintainability of web application (analysability, changeability, stability)</td>
<td>Navigational models</td>
<td>distance framework valid, shown</td>
<td>spearman</td>
<td>Academic exp</td>
</tr>
<tr>
<td>---Abrahaо et al. [80]</td>
<td>functional size (OO-method function points for the web)</td>
<td>efficiency and effectiveness (measurement time, reproducibility), perceived easy of use, usefulness and intention to use</td>
<td>web applications</td>
<td>no</td>
<td>one tailed t-test</td>
<td>academic experiment</td>
</tr>
<tr>
<td>---Abran and rabillard [81]</td>
<td>function points</td>
<td>work-effort</td>
<td>software application</td>
<td>analysis of FP (scale perspective)</td>
<td>ols</td>
<td>37 indust projects</td>
</tr>
<tr>
<td>---Alagar et al. [82]</td>
<td>architectural complexity measures</td>
<td>complexity, reliability</td>
<td>maintenance process</td>
<td>representational measure theory</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>---Allen [83]</td>
<td>measures of size, length, complexity, coupling, cohesion</td>
<td>sw system in form of a graph</td>
<td>property based and kitch et al.</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>---almeida et al. [84]</td>
<td>ASAP product measures (number of operands, operators, declarative, nline comments, distinct operators, blank lines........</td>
<td>effort to isolate and correct a faulty component</td>
<td>Ada software components</td>
<td>no</td>
<td>ML alg (newID, CN2, C4.5, FOIL)</td>
<td>GSS reuse asset lib (NASA)</td>
</tr>
<tr>
<td>--- Alshayeb and Li [85]</td>
<td>Object oriented metrics</td>
<td>LOC added, deleted, changed, maintenance and refactoring effort</td>
<td>Short-long cycled processes</td>
<td>no</td>
<td>multiple linear regression</td>
<td>industrial dataset, lab analysis</td>
</tr>
<tr>
<td>Reference</td>
<td>Measures</td>
<td>Attribute</td>
<td>Entity</td>
<td>Theoretic</td>
<td>Empirical</td>
<td>Environment</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>-----------</td>
<td>--------</td>
<td>-----------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>--- Ambriola Gervasi [86]</td>
<td>measures of stability (amount of info contained in req at time t) and efficiency</td>
<td>amount of rework, perceived work efficiency</td>
<td>Requirements analysis process</td>
<td>no</td>
<td>no real analysis, plot of data</td>
<td>Academic experim</td>
</tr>
<tr>
<td>--- Antoniol et al.[87]</td>
<td>classes, methods, associations, inheritance, a combination of them, OOP-related measures</td>
<td>size</td>
<td>oo software</td>
<td>no</td>
<td>linear regr. [simple]</td>
<td>4 indust projects</td>
</tr>
<tr>
<td>--- Arisholm et al. [88]</td>
<td>size, static coupling, 12 dynamic coupling measures (import and export coupling)</td>
<td>change proneness</td>
<td>oo software</td>
<td>property based</td>
<td>pca, uni/multi ols,</td>
<td>large open source java system</td>
</tr>
<tr>
<td>--- Bohli Rivard [89]</td>
<td>14 measures of risk factors (see table 2 of the paper)</td>
<td>risk factors</td>
<td>Information Technology outsourcing</td>
<td>no</td>
<td>expert opinion (not clear), PLS No correlation study</td>
<td>db of 3000 ind. orgs.</td>
</tr>
<tr>
<td>--- Bandi, et al. [90]</td>
<td>design complexity, maint task, programmer ability (interaction level, interface size, and operation argument complexity)</td>
<td>maintenance performance (time)</td>
<td>Object Oriented Design</td>
<td>previously val with weyuk prop</td>
<td>ANOVA, correlation, linear regression</td>
<td>Academic</td>
</tr>
<tr>
<td>--- Barnard [91]</td>
<td>CK set, loc, methods, attributes, meaningfulness of variable name</td>
<td>subjective perception of reusability</td>
<td>classes (oo code)</td>
<td>no</td>
<td>ad hoc comparision</td>
<td>acad exp with exper delev, exp 2 classes from know soft</td>
</tr>
<tr>
<td>--- Basili, et al. [92]</td>
<td>adjusted measures of CK suite (Weighted methods per class, depth of inheritance tree, #children of a class, coupling between object classes, response for a class, lack of cohesion on methods), code metrics nesting level, func def, func call.</td>
<td>Fault proneness</td>
<td>Object Oriented Design classes</td>
<td>perf in [21](axiom B) and [163] fen ton (meas theory) e weyu</td>
<td>uni/multi variate logistic regression</td>
<td>8 c++ inf manag systems Academic</td>
</tr>
</tbody>
</table>

**TABLE 1:**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretic</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>--- Ambriola Gervasi [86]</td>
<td>measures of stability (amount of info contained in req at time t) and efficiency</td>
<td>amount of rework, perceived work efficiency</td>
<td>Requirements analysis process</td>
<td>no</td>
<td>no real analysis, plot of data</td>
<td>Academic experim</td>
</tr>
<tr>
<td>--- Antoniol et al.[87]</td>
<td>classes, methods, associations, inheritance, a combination of them, OOP-related measures</td>
<td>size</td>
<td>oo software</td>
<td>no</td>
<td>linear regr. [simple]</td>
<td>4 indust projects</td>
</tr>
<tr>
<td>--- Arisholm et al. [88]</td>
<td>size, static coupling, 12 dynamic coupling measures (import and export coupling)</td>
<td>change proneness</td>
<td>oo software</td>
<td>property based</td>
<td>pca, uni/multi ols,</td>
<td>large open source java system</td>
</tr>
<tr>
<td>--- Bohli Rivard [89]</td>
<td>14 measures of risk factors (see table 2 of the paper)</td>
<td>risk factors</td>
<td>Information Technology outsourcing</td>
<td>no</td>
<td>expert opinion (not clear), PLS No correlation study</td>
<td>db of 3000 ind. orgs.</td>
</tr>
<tr>
<td>--- Bandi, et al. [90]</td>
<td>design complexity, maint task, programmer ability (interaction level, interface size, and operation argument complexity)</td>
<td>maintenance performance (time)</td>
<td>Object Oriented Design</td>
<td>previously val with weyuk prop</td>
<td>ANOVA, correlation, linear regression</td>
<td>Academic</td>
</tr>
<tr>
<td>--- Barnard [91]</td>
<td>CK set, loc, methods, attributes, meaningfulness of variable name</td>
<td>subjective perception of reusability</td>
<td>classes (oo code)</td>
<td>no</td>
<td>ad hoc comparision</td>
<td>acad exp with exper delev, exp 2 classes from know soft</td>
</tr>
<tr>
<td>--- Basili, et al. [92]</td>
<td>adjusted measures of CK suite (Weighted methods per class, depth of inheritance tree, #children of a class, coupling between object classes, response for a class, lack of cohesion on methods), code metrics nesting level, func def, func call.</td>
<td>Fault proneness</td>
<td>Object Oriented Design classes</td>
<td>perf in [21](axiom B) and [163] fen ton (meas theory) e weyu</td>
<td>uni/multi variate logistic regression</td>
<td>8 c++ inf manag systems Academic</td>
</tr>
</tbody>
</table>
### TABLE 1:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretical</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baudry et al. [93]</td>
<td>robustness and diagnosability measures</td>
<td>oo systems designed by contracts</td>
<td>axiomatic approach of shepherd 93</td>
<td>applic of meas. to 3 systems</td>
<td>3 industrial case studies</td>
<td></td>
</tr>
<tr>
<td>Benlarbi et al. [94]</td>
<td>CK without LCOM (they say that ck set is the most referred)</td>
<td>Fault proneness classes</td>
<td>not mentioned, but maybe the CK has been theor val.</td>
<td>univ log regression</td>
<td>commercial applic c++ teller-communic system</td>
<td></td>
</tr>
<tr>
<td>Benlarbi and Melo [95]</td>
<td>measures of polymorphism based on compile time linking decisions and based on runtime binding decisions</td>
<td>Fault proneness Object Oriented Design classes</td>
<td>no</td>
<td>uni/multivariate logistic regression</td>
<td>industrial oo system</td>
<td></td>
</tr>
<tr>
<td>Bie-man et al. [96]</td>
<td>patterns, design attributes, class size (total number of attributes, operations, number of friends methods, of methods that are overridden, depth of inheritance, number of direct child classes, number of descendants.</td>
<td>number of changes oo classes</td>
<td>discuss of construct val only for the dep variab.</td>
<td>spearman, pearson, linear regr.</td>
<td>industrial oo system</td>
<td></td>
</tr>
<tr>
<td>Binley Shach ice [97]</td>
<td>coupling metrics</td>
<td>run-time failures, maintenance time, faults due to maint.act</td>
<td>module #?</td>
<td>spearman correlation</td>
<td>industrial 4-ind case studies</td>
<td></td>
</tr>
<tr>
<td>Binley et al. [98]</td>
<td>16 metrics: coupling, inheritance tree, size of response data abstraction, permitted interaction, coupling dependency quality: impact on predicted implementation, maintenance effort oo design</td>
<td>no</td>
<td>expert opinion evaluation</td>
<td>industrial experts, bad paper</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Briand Bunse bbd97 [99]</td>
<td>procedural vs. OO design; adherence to common principles of good design (Coupling, cohesion, clarity of design, generalisation/specialization depth, keeping object and classes simple) understandability (Que_%), effectiveness of modification (Mod. %) modification rate of Impact Analysis (Mod_rate) oo and structured design documents discussion in construct val. 2x2 fact. Design; ANOVA, paired t-test</td>
<td>time and correctness of understandability, time, completeness, correctness and rate of modification oo design</td>
<td>discussion in construct val.</td>
<td>academ exp, replication of previous</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Briand Bunse bbd97 [101]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 1:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretical</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Briand, Daly, Wuest [103]</td>
<td>Measures of coupling, cohesion inheritance</td>
<td>probability of Fault detection</td>
<td>Object Oriented Design system classes</td>
<td>perform in [108], [104] (except inher meas)</td>
<td>uni/ multivariate logistic regr.</td>
<td>Academic</td>
</tr>
<tr>
<td>Briand et al. [102] issue 98</td>
<td>several kinds of LCOM (lack of cohesion in methods), connectivity, tight and loose class cohesion, information flow based cohesion, ratio of cohesive interaction (RCI), neutral, pessimistic and optimistic RCI</td>
<td>oo systems</td>
<td>briand property based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Briand, Daly, Wuest [104]</td>
<td>coupling between objects (CBO), response for classe (RFC), message passing coupling (MPC), data abstraction coupling DAC, efferent and afferent coupling (CE CA), coupling factor COF, information flow based coupling ICP; +coupling measures defined in [107]</td>
<td>oo systems</td>
<td>briand property based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Briand et al. [105] metrics 98</td>
<td>Measures of coupling, cohesion inheritance</td>
<td>probability of Fault detection</td>
<td>Object Oriented Design system classes</td>
<td>performed in [108], [104]</td>
<td>pca, univariate logistic regr.</td>
<td>Academic exp</td>
</tr>
<tr>
<td>Briand wust et al. [106]</td>
<td>all the design measures proposed in the literature up to 1999 including CK and CFOOD (28coupling, 10cohesion 1 i inheritance), size measures ~50</td>
<td>fault proneness</td>
<td>oo system classes</td>
<td>partial val, performed in [108], [104], inh meas not validated</td>
<td>pca, uni/multivariate logistic regr.</td>
<td>Industrial case study</td>
</tr>
<tr>
<td>Briand et al. [107] icse 97</td>
<td>relationship, locus and type coupling measures (the last can be: class-attribute, class-method, method-method interactions)</td>
<td>fault proneness</td>
<td>c++ classes</td>
<td>measures are defined formally, this is their valid.</td>
<td>logistic regression</td>
<td>Academic case study</td>
</tr>
<tr>
<td>Briand melo wust [108]</td>
<td>coupling, polymorphism measures, subset of CK measures and size measures (22 measures in total)</td>
<td>Fault proness</td>
<td>oo design</td>
<td>no</td>
<td>uni/multivariate logistic regr. multivariate adaptive regr splines</td>
<td>Industrial sw systems</td>
</tr>
<tr>
<td>Briand Morasca [109]</td>
<td>Axiom size measure and item size measures</td>
<td>Quality indicators of specification change and effort</td>
<td>Formal specification language (TRIO+)</td>
<td>no</td>
<td>uni/ multivariate logistic/ linear regr</td>
<td>Industrial case study</td>
</tr>
<tr>
<td>Reference</td>
<td>Measures</td>
<td>Attribute</td>
<td>Entity</td>
<td>Theoretical Validation</td>
<td>Empirical Validation</td>
<td>Environment</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-------------------------------------</td>
<td>------------------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Briand, et al. [110]</td>
<td>Interaction based measure for cohesion and coupling, measures based on USES and IS, COMPO-NENT_OF relationship</td>
<td>Fault proneness</td>
<td>Object based high level design (their own)</td>
<td>pearson, uni/multivariate logistic regression</td>
<td>real scale industrial case study</td>
<td></td>
</tr>
<tr>
<td>Briand wust BW01 [111], [112]</td>
<td>size (classes, attributes and methods), cohesion (relationship between methods and class attributes), coupling, and inheritance (relationship between classes, and between methods) complexity, (~50 measures including CK and CFOOD)</td>
<td>development effort</td>
<td>class (source code) for coupling and cohesion measures performed in [103] and [104]</td>
<td>negative binomial, Poisson Regression with CART regression trees</td>
<td>c++ system, university setting</td>
<td></td>
</tr>
<tr>
<td>Briand wust et al. [113] icse 99</td>
<td>28 Measures of coupling, 10 cohesion</td>
<td>fault proneness</td>
<td>oo design system classes</td>
<td>performed in [108] [104]</td>
<td>pca, uni/multivariate logistic regression</td>
<td>industrial system by professionals</td>
</tr>
<tr>
<td>Briand et al. bwl 99 [114]</td>
<td>measures from [104] + 3 more measures based on counting direct and indirect method invocation and aggregation relationships</td>
<td>likelihood of ripple changes</td>
<td>oo systems</td>
<td>no</td>
<td>PCA, LR ranking-based model</td>
<td>commercial c++ system</td>
</tr>
<tr>
<td>Briand et al. bwl 01 [115]</td>
<td>~50 measures including CK and CFOOD</td>
<td>fault proneness</td>
<td>oo systems</td>
<td>property based port perf in briand 98 e 99</td>
<td>uni/multivariate regression</td>
<td>commercial sys</td>
</tr>
<tr>
<td>Brito-Abreu Melo [116]</td>
<td>mood set</td>
<td>maintainability (normalised rework) reliability (defect density) fault density</td>
<td>oo design</td>
<td>no</td>
<td>pearson, multiple linear regression</td>
<td>academic, correlat</td>
</tr>
<tr>
<td>Calero et al. [117] AICCSA</td>
<td>num of tables in a db schema, num of attributes, num of foreign keys</td>
<td>analysability, changeability, stability, testability</td>
<td>database schema</td>
<td>performed in [] (meas theory)and in [] (prop based)</td>
<td>spearman statistic</td>
<td>4 large dbs from research centre</td>
</tr>
<tr>
<td>Calero et al. [118]</td>
<td>referential integrity metrics (num of foreign keys, depth of rel tree)</td>
<td>analysability,</td>
<td>relational database</td>
<td>perf in [] (use) and [] (briand)</td>
<td>F statistic</td>
<td>industrial exp</td>
</tr>
</tbody>
</table>

**TABLE 1:**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretical Validation</th>
<th>Empirical Validation</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Briand, et al. [110]</td>
<td>Interaction based measure for cohesion and coupling, measures based on USES and IS, COMPO-NENT_OF relationship</td>
<td>Fault proneness</td>
<td>Object based high level design (their own)</td>
<td>pearson, uni/multivariate logistic regression</td>
<td>real scale industrial case study</td>
<td></td>
</tr>
<tr>
<td>Briand wust BW01 [111], [112]</td>
<td>size (classes, attributes and methods), cohesion (relationship between methods and class attributes), coupling, and inheritance (relationship between classes, and between methods) complexity, (~50 measures including CK and CFOOD)</td>
<td>development effort</td>
<td>class (source code) for coupling and cohesion measures performed in [103] and [104]</td>
<td>negative binomial, Poisson Regression with CART regression trees</td>
<td>c++ system, university setting</td>
<td></td>
</tr>
<tr>
<td>Briand wust et al. [113] icse 99</td>
<td>28 Measures of coupling, 10 cohesion</td>
<td>fault proneness</td>
<td>oo design system classes</td>
<td>performed in [108] [104]</td>
<td>pca, uni/multivariate logistic regression</td>
<td>industrial system by professionals</td>
</tr>
<tr>
<td>Briand et al. bwl 99 [114]</td>
<td>measures from [104] + 3 more measures based on counting direct and indirect method invocation and aggregation relationships</td>
<td>likelihood of ripple changes</td>
<td>oo systems</td>
<td>no</td>
<td>PCA, LR ranking-based model</td>
<td>commercial c++ system</td>
</tr>
<tr>
<td>Briand et al. bwl 01 [115]</td>
<td>~50 measures including CK and CFOOD</td>
<td>fault proneness</td>
<td>oo systems</td>
<td>property based port perf in briand 98 e 99</td>
<td>uni/multivariate regression</td>
<td>commercial sys</td>
</tr>
<tr>
<td>Brito-Abreu Melo [116]</td>
<td>mood set</td>
<td>maintainability (normalised rework) reliability (defect density) fault density</td>
<td>oo design</td>
<td>no</td>
<td>pearson, multiple linear regression</td>
<td>academic, correlat</td>
</tr>
<tr>
<td>Calero et al. [117] AICCSA</td>
<td>num of tables in a db schema, num of attributes, num of foreign keys</td>
<td>analysability, changeability, stability, testability</td>
<td>database schema</td>
<td>performed in [] (meas theory)and in [] (prop based)</td>
<td>spearman statistic</td>
<td>4 large dbs from research centre</td>
</tr>
<tr>
<td>Calero et al. [118]</td>
<td>referential integrity metrics (num of foreign keys, depth of rel tree)</td>
<td>analysability,</td>
<td>relational database</td>
<td>perf in [] (use) and [] (briand)</td>
<td>F statistic</td>
<td>industrial exp</td>
</tr>
<tr>
<td>Reference</td>
<td>Measures</td>
<td>Attribute</td>
<td>Entity</td>
<td>Theoretical</td>
<td>Empirical</td>
<td>Environ</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>----------------</td>
<td>-------------</td>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>---calero et al. [119]</td>
<td>table, star, schema,</td>
<td>----</td>
<td>data warehouse</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>---calero et al. [120]</td>
<td>table size, number of involved classes, number of shared classes, percentage of complex columns</td>
<td>complexity</td>
<td>object relational database</td>
<td>some discussion in construction</td>
<td>pearson, spearman</td>
<td>academic exp</td>
</tr>
<tr>
<td>---calero et al. [121]</td>
<td>the ones above + referential degree (number of foreign keys) depth of relational tree</td>
<td>understandability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---confora et al. [122]</td>
<td>structural properties measures (size, complexity, coupling)</td>
<td>maintainability</td>
<td>process models</td>
<td>poels ded applied but not shown</td>
<td>spearman</td>
<td></td>
</tr>
<tr>
<td>---cartwright shepperd [123]</td>
<td>13 measures (depth of inheritance tree, num of children, attributes, states, events, reads, writes, deletes, locs, defects)</td>
<td>defect proneness, size</td>
<td>oo class?</td>
<td>no</td>
<td>linear regression (uni-muli)</td>
<td>industrial system</td>
</tr>
<tr>
<td>---chidamb er et al. [124]</td>
<td>CK suite</td>
<td>productivity, reuse effort, design effort</td>
<td>oo design classes</td>
<td>no</td>
<td>linear stepwise reg</td>
<td>3 oo systems</td>
</tr>
<tr>
<td>---chidamb er kemerer [125]</td>
<td>6 design metrics (CK) [Weighted methods per class, depth of inheritance tree, #children of a class, coupling between object classes, response for a class, lack of cohesion on methods]</td>
<td>---</td>
<td>oo design</td>
<td>weyuker</td>
<td>only theoretical, descriptive stat.</td>
<td>2 industrial projects</td>
</tr>
<tr>
<td>---chidamb er kemerer [126]</td>
<td>size and complexity</td>
<td></td>
<td>oo design</td>
<td>weyuker</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>---cos-tagliola et al. [127]</td>
<td>class point measures</td>
<td>development effort</td>
<td>oo products</td>
<td>briand properties</td>
<td>univariate ols</td>
<td>40 stud projects</td>
</tr>
<tr>
<td>---cos-tagliola et al [128]</td>
<td>class point</td>
<td>size in LOC</td>
<td>oo gui</td>
<td>briand properties</td>
<td>uni ols</td>
<td>35 java study systems</td>
</tr>
<tr>
<td>Reference</td>
<td>Measures</td>
<td>Attribute</td>
<td>Entity</td>
<td>Theoretical</td>
<td>Empirical</td>
<td>Environment</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------</td>
<td>---------------</td>
<td>----------------</td>
<td>-------------</td>
<td>----------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>---dagpinar et al. [129]</td>
<td>Measures of size, coupling, cohesion</td>
<td>maintainability</td>
<td>oo classes</td>
<td>no</td>
<td>uni/multivariate</td>
<td>academic sw sys</td>
</tr>
<tr>
<td></td>
<td>inheritance</td>
<td></td>
<td></td>
<td></td>
<td>linear regr.</td>
<td>tem</td>
</tr>
<tr>
<td>---delucia et al. [130]</td>
<td>size, number of maintenance tasks split in 3 categories</td>
<td>effort</td>
<td>sw maintenance process</td>
<td>no</td>
<td>multivariate linear regr.</td>
<td>industrial</td>
</tr>
<tr>
<td>---devambu et al. [131]</td>
<td>reuse level, reuse frequency, size and frequency, reused source instructions</td>
<td>productivity, defect density, rework effort</td>
<td>software reuse</td>
<td>modficiation of weyukers</td>
<td>correlation, OLS</td>
<td>academic</td>
</tr>
<tr>
<td>---diaz et al [132]</td>
<td>trigger complexity (triggering potential, number of anchors, distance of a trigger)</td>
<td>understandability (measured with time)</td>
<td>active DBMS</td>
<td>zuse framework [214214]</td>
<td>F-statistic</td>
<td>mix acad industr exp</td>
</tr>
<tr>
<td>---El Emam, et al. [133]</td>
<td>CK suite, and a subset of Loretz and Kidd metrics (18 in total)</td>
<td>fault proneness and class size (confounding effect of size)</td>
<td>oo design classes</td>
<td>perf in [108] [104] [125]</td>
<td>spearman logistic regression</td>
<td>industrial sw</td>
</tr>
<tr>
<td>---el emam ebml00 [134]</td>
<td>Stmts (number of declaration and executable statements in the methods), number of methods, number of Attributes,</td>
<td>fault proneness</td>
<td>c++ classes</td>
<td>no</td>
<td>univ. logistic regression</td>
<td>2 c++ sys, 1 java industr</td>
</tr>
<tr>
<td>---el emam ebg99r [135]</td>
<td>24 metrics: (C&amp;K, and briand et al. [107]); only 4 of them are correlated.</td>
<td>fault proneness</td>
<td>c++ classes</td>
<td>no</td>
<td>univ/multi log regr.</td>
<td>telemcomm c++</td>
</tr>
<tr>
<td>---el emam emm01 [137]</td>
<td>10 measures, subset of C and briand [part of C-Food, DIT, NOC, Attras]</td>
<td>fault proneness</td>
<td>java classes</td>
<td>no</td>
<td>univ/multi log regr.</td>
<td>commercial java appl.</td>
</tr>
<tr>
<td>---ferandez dolado [138]</td>
<td>MACOVs and MACOVs: space and time [num of instructions] and [binary storage]</td>
<td>algorithmic cost of verification</td>
<td>briand property based</td>
<td>difficult: logical consequences of relationships in the models presented</td>
<td>2 comm syst, 3 acad cas study</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 1:**

Reference Measures Attribute Entity Theoretic Empirical Environment
<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretic al</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>---Ferrell et al.</td>
<td>AIMS set (automated information flow measurement set) based on coupling and control flow</td>
<td>corrective and preventive maintenance (error rate)</td>
<td>design stage of software development</td>
<td>no</td>
<td>spearman</td>
<td>medium size ind proj</td>
</tr>
<tr>
<td>---Horavantesi et al.</td>
<td>226 oo metrics</td>
<td>fault proneness</td>
<td>classes</td>
<td>no</td>
<td>PCA multi log regr</td>
<td>8 acad system</td>
</tr>
<tr>
<td>---Garofolo et al.</td>
<td>summary</td>
<td>class complexity</td>
<td>development and maintenance effort</td>
<td>oo system</td>
<td>axiomatic (briand)</td>
<td>linear regression</td>
</tr>
<tr>
<td>---Garcia et al.</td>
<td>structural properties measures (size, complexity, coupling)</td>
<td>maintainability (understandability, modifiability)</td>
<td>process models</td>
<td>no, linear regression</td>
<td>spearman, acadet</td>
<td>acad, experim</td>
</tr>
<tr>
<td>---Garcia et al.</td>
<td>Structural complexity metrics (entity, attributes and relationship metrics)</td>
<td>maintainability (analyzability, understandability, modifiability)</td>
<td>Conceptual data model (ERD)</td>
<td>some discussion in constr valid</td>
<td>spearman, acade</td>
<td>acad, experim</td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>Structural complexity metrics</td>
<td>maintainability (analyzability, understandability, modifiability)</td>
<td>Class diagram</td>
<td>perf in [118] with poels ded.</td>
<td>spearman, Academ</td>
<td></td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>6 measures of structural complexity</td>
<td>Understandability, modifiability</td>
<td>Conceptual data model (ERD)</td>
<td>performed in [143] with poels ded.</td>
<td>pearson, Academ exp</td>
<td></td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>structural complexity metrics (num of: classes, attributes, methods, associations, aggregation, dependencies, generalisation, generalisation hierarchies, maximum depth of the generalization hierarchies, maximum height of the aggregation hierarchies.)</td>
<td>understandability, modifiability</td>
<td>UML Class diagram</td>
<td>no</td>
<td>pearson multi ols, acad, exp</td>
<td></td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>structural complexity metrics</td>
<td>complexity</td>
<td>UML class diagrams</td>
<td>discussion in construct val.</td>
<td>fuzzy rules method, acad, exp</td>
<td></td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>maintenance time</td>
<td>UML class diagram</td>
<td>perf in [143] with poels edene</td>
<td>spearman, acad, exp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 1:**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretic al</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>---Ferrell et al.</td>
<td>AIMS set (automated information flow measurement set) based on coupling and control flow</td>
<td>corrective and preventive maintenance (error rate)</td>
<td>design stage of software development</td>
<td>no</td>
<td>spearman</td>
<td>medium size ind proj</td>
</tr>
<tr>
<td>---Horavantesi et al.</td>
<td>226 oo metrics</td>
<td>fault proneness</td>
<td>classes</td>
<td>no</td>
<td>PCA multi log regr</td>
<td>8 acad system</td>
</tr>
<tr>
<td>---Garofolo et al.</td>
<td>summary</td>
<td>class complexity</td>
<td>development and maintenance effort</td>
<td>oo system</td>
<td>axiomatic (briand)</td>
<td>linear regression</td>
</tr>
<tr>
<td>---Garcia et al.</td>
<td>structural properties measures (size, complexity, coupling)</td>
<td>maintainability (understandability, modifiability)</td>
<td>process models</td>
<td>no, linear regression</td>
<td>spearman, acadet</td>
<td>acad, experim</td>
</tr>
<tr>
<td>---Garcia et al.</td>
<td>Structural complexity metrics (entity, attributes and relationship metrics)</td>
<td>maintainability (analyzability, understandability, modifiability)</td>
<td>Conceptual data model (ERD)</td>
<td>some discussion in constr valid</td>
<td>spearman, acade</td>
<td>acad, experim</td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>Structural complexity metrics</td>
<td>maintainability (analyzability, understandability, modifiability)</td>
<td>Class diagram</td>
<td>perf in [118] with poels ded.</td>
<td>spearman, Academ</td>
<td></td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>6 measures of structural complexity</td>
<td>Understandability, modifiability</td>
<td>Conceptual data model (ERD)</td>
<td>performed in [143] with poels ded.</td>
<td>pearson, Academ exp</td>
<td></td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>structural complexity metrics (num of: classes, attributes, methods, associations, aggregation, dependencies, generalisation, generalisation hierarchies, maximum depth of the generalization hierarchies, maximum height of the aggregation hierarchies.)</td>
<td>understandability, modifiability</td>
<td>UML Class diagram</td>
<td>no</td>
<td>pearson multi ols, acad, exp</td>
<td></td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>structural complexity metrics</td>
<td>complexity</td>
<td>UML class diagrams</td>
<td>discussion in construct val.</td>
<td>fuzzy rules method, acad, exp</td>
<td></td>
</tr>
<tr>
<td>---Genero et al.</td>
<td>maintenance time</td>
<td>UML class diagram</td>
<td>perf in [143] with poels edene</td>
<td>spearman, acad, exp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Measures</td>
<td>Attribute</td>
<td>Entity</td>
<td>Theoretical</td>
<td>Empirical</td>
<td>Environmental</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>-----------</td>
<td>--------</td>
<td>-------------</td>
<td>-----------</td>
<td>---------------</td>
</tr>
<tr>
<td>---gen-ero et al. [151] metrics 2003</td>
<td>6 structural complexity metrics, 3 size metrics</td>
<td>maintainability (understandability time, modifiability time, correctness and completeness)</td>
<td>UML class diagram</td>
<td>performed in [143] (do not say how)</td>
<td>pca, univariate linear regr.</td>
<td>2 academic studies</td>
</tr>
<tr>
<td>---glosberg gemm00 [152]</td>
<td>subset of CK and Briand [107] (DIT, NOC, Mthds, part of C-Food)</td>
<td>fault proneness</td>
<td>java classes</td>
<td>other (cognitive theory)</td>
<td>uni/multilogist regr.</td>
<td>xpose, commercial java</td>
</tr>
<tr>
<td>---gursaran [153]</td>
<td>DIT, NOC, (2 metrics from CK)</td>
<td>---</td>
<td>c++ classes</td>
<td>representational theory</td>
<td>---</td>
<td>academic</td>
</tr>
<tr>
<td>---gursaran ray [154]</td>
<td>inheritance metrics from CK and from britoAbreu and corapuca</td>
<td>complexity</td>
<td>oo classes</td>
<td>weyuker (prop 9)</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>---gyimothy et al [155]</td>
<td>CK</td>
<td>fault proneness</td>
<td>open source software</td>
<td>no</td>
<td>multi log and linear, regression and ML</td>
<td>Mozilla source code</td>
</tr>
<tr>
<td>---hall et al. [156]</td>
<td>parameter [size, structure, computation uses, predicate uses], passed arguments, global variables (Glbv), Glbv (changed, computation uses, predicate uses, function (calls, retums, void uses) returned structures + 15 size metrics</td>
<td>coupling complexity (size of interface, type of information flow, type of passed data, global connection, interaction with other modules)</td>
<td>design modules</td>
<td>no</td>
<td>pca</td>
<td>clone of the unix &quot;spell&quot; utility</td>
</tr>
<tr>
<td>---harrison h98 [157]</td>
<td>Ck inheritance (DIT, NOC), NMO, NMI</td>
<td>fault density, non comment source lines, software understanding (Subjective complexity)</td>
<td>c++ classes</td>
<td>no</td>
<td>spearman rho</td>
<td>mix indus acad systems</td>
</tr>
<tr>
<td>---harrison hcn00 [158]</td>
<td>DIT (flat vs deep inheritance structure)</td>
<td>maintainability, understandability</td>
<td>oo software</td>
<td>no</td>
<td>4x12 betw subject; chi square</td>
<td>academ exp 2 c++ systems</td>
</tr>
<tr>
<td>Reference</td>
<td>Measures</td>
<td>Attribute</td>
<td>Entity</td>
<td>Theoretical</td>
<td>Empirical</td>
<td>Environment</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
<td>-----------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>---harrison et al. [159]</td>
<td>MOOD set (6 metrics) (both, this should be the same as ncn98)</td>
<td>inheritance, coupling, encapsulation, polymorphism</td>
<td>oo classes</td>
<td>kitch approach [39]</td>
<td>descriptive data only</td>
<td>commercial software</td>
</tr>
<tr>
<td>---harrison et al. [160]</td>
<td>coupling between objects, number of associations</td>
<td>understandability, num errors, error density</td>
<td>oo class design</td>
<td>no</td>
<td>spearman</td>
<td>S system</td>
</tr>
<tr>
<td>---harrison hsd96 [161]</td>
<td>non comment source lines, lib/non-lib functions called, depth in call graph, #function declarations /definitions, #domain specific function</td>
<td>Subjective complexity, #faults in testing, time to fix faults #modification requests, time to modify</td>
<td>oo source code</td>
<td>no</td>
<td>Spearman, Kendall, Pearson's r for all pairs</td>
<td>C++ system</td>
</tr>
<tr>
<td>---hast and sajeve [162]</td>
<td>Functionality (num of atomic units in the signature section of the spec) problem complexity (num of atomic units in the semantic section of the specification)</td>
<td>size</td>
<td>soft (specified with algebraic specification language)</td>
<td>kitch, pfleeger etc framework</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>---hitz and montazeri [163]</td>
<td>ck</td>
<td>oo design</td>
<td>Represenation condition [baker] [fenton] and properties wey-ucker</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>---Idri abran [164]</td>
<td>fuzzy logic measures</td>
<td>similarity</td>
<td>software project</td>
<td>axiomatic (fenton?) (ifpug)</td>
<td>descriptive data compact</td>
<td>cocomo projects used</td>
</tr>
<tr>
<td>---Idri abran [165]</td>
<td>fuzzy logic measures (distance between projects) based on numerical and linguistic values</td>
<td>project effort</td>
<td>sw development</td>
<td>done in previous</td>
<td>fuzzy analogy</td>
<td>cocomo 81 dataset</td>
</tr>
<tr>
<td>---ivory et al [166]</td>
<td>page composition (word, link and graphic count) page formatting (emphasized text, text positioning, text clusters) overall page characteristics (page size, download speed)</td>
<td>content, structure &amp;navigation, visual design, functionality, interactivity and overall experience (judge rates)</td>
<td>web page design</td>
<td>---</td>
<td>linear reg, linear discriminant analysis, t-test</td>
<td>1898 pages from the webby awards categ</td>
</tr>
</tbody>
</table>

TABLE 1:
TABLE 1:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretic</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jorgensen [167]</td>
<td>cause of task, change degree on code, type of operation on code and confidence on maintainer</td>
<td>maintenance task effort</td>
<td>maintenance process?</td>
<td>no</td>
<td>linear regression</td>
<td>industrial</td>
</tr>
<tr>
<td>Jung et al. [168]</td>
<td>LOC, Halstead Software Science (num of unique operators and operands, total num of operators and operands), McCabe’s cyclomatic num, num of signals, num of database accesses, num of library calls.</td>
<td>number of change requests due to faults in testing</td>
<td>modules in testing</td>
<td>no</td>
<td>negative binomial regression</td>
<td>large scale telecom sys (816 modules in Chill)</td>
</tr>
<tr>
<td>Khoshgoftaar and Allen [169]</td>
<td>modules used, total and unique calls, if then conditional arcs, loops, nesting level, span of conditional arcs, span of loops, McCabe cyclomatic complexity, indicator of reuse from the prior release and prior prototypes, metrics of development process prior to integration</td>
<td>fault proneness</td>
<td>module</td>
<td>no</td>
<td>non parametric discriminant analysis, pca, univariate regression</td>
<td>2 industrial case studies</td>
</tr>
<tr>
<td>Koru et al [170]</td>
<td>static measures of size; class and method level measures for one of the products</td>
<td>defect</td>
<td>modules and classes</td>
<td>no</td>
<td>ML algorithms (j48 and KStar)</td>
<td>5 nasa products</td>
</tr>
<tr>
<td>Lakshmanan et al. [171]</td>
<td>cyclomatic number, total adjusted complexity, scope ratio, npath, mebow.</td>
<td>complexity</td>
<td>control flow (source code)</td>
<td>weyuker</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Li &amp; Henry [172]</td>
<td>3 CK measures (out of 6). 3 coupling measures, a class interface increment and 2 size measures</td>
<td>maintenance effort (number of lines changed per class)</td>
<td>oo systems</td>
<td>no</td>
<td>multi linear regression</td>
<td>2 commercial systems</td>
</tr>
<tr>
<td>Li Henry [173]</td>
<td>C&amp;K - CBO, MPC, DAC, 2xSize (Stmts, Mth+Attr)</td>
<td>maintenance effort surrogate (should be the same as previous)</td>
<td>oo classes</td>
<td>no</td>
<td>multi linear regression</td>
<td>2 ada commercial sys</td>
</tr>
<tr>
<td>Ioconoole int rep [174]</td>
<td>10 measures</td>
<td>size, ....</td>
<td>requirements management</td>
<td>measure theory kitchen framework</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Reference</td>
<td>Measures</td>
<td>Attribute</td>
<td>Entity</td>
<td>Theoretic al</td>
<td>Empirical</td>
<td>Environment</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------</td>
<td>------------------------------------------------</td>
<td>----------------------</td>
<td>---------------</td>
<td>-------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Ioconsoli et al. [175] [176]</td>
<td>num lines, words, actors, use cases, subjects opinion</td>
<td>volatility (num changes, max mode, min, revisions)</td>
<td>requirements</td>
<td>previously performed</td>
<td>spearman</td>
<td>industrial sw</td>
</tr>
<tr>
<td></td>
<td>num lines, words, actors, use cases,</td>
<td>volatility (num changes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>5 functional specifications metrics, 3 data models metrics</td>
<td>project size, development effort</td>
<td>4GL database systems</td>
<td>no</td>
<td>pearson, spearman, stepwise linear regress</td>
<td>74sytems senior students</td>
</tr>
<tr>
<td>---</td>
<td>8 structural complexity and 3 size metrics some set used in [148])</td>
<td>maintainability (analyzability, understandability, modifiability)</td>
<td>uml class diagrams</td>
<td>performed in [143] poels and dedene</td>
<td>pca, spearman, pearson</td>
<td>3 academ exper</td>
</tr>
<tr>
<td>---</td>
<td>path metrics (compacteness, stratum)</td>
<td>performance in hypertext search task (print ability, computer experience, pages viewed, order of path matrix)</td>
<td>hypertext network</td>
<td>no</td>
<td>pearson corr coeff, multiple linear regr.</td>
<td>2 academ studies</td>
</tr>
<tr>
<td>---</td>
<td>hyperdocument size, connectivity, perceived compactness and stratum. (3 out of 4 are valid)</td>
<td>reusability, maintainability, development effort (link generality)</td>
<td>web applications</td>
<td>represenational theory [kitch]</td>
<td>spearman</td>
<td>academ case study</td>
</tr>
<tr>
<td>---</td>
<td>highlighting of anchors, representation and type of links, size and structure of the application, connectivity, compactness, stratum, , role and experience of the author</td>
<td>understandability (structural complexity)</td>
<td>educational hypermedia applications</td>
<td>NO but gam</td>
<td>gamma correlation</td>
<td>academic, survey</td>
</tr>
<tr>
<td>---</td>
<td>num of transition, num of states, num of activities .... num of entry and exit actions.</td>
<td>understandability (structural complexity)</td>
<td>behavioral diagrams (uml state-charts)</td>
<td>previously perf with poels and ded in [143]</td>
<td>spearman</td>
<td>acad exp replication of [143]</td>
</tr>
<tr>
<td>---</td>
<td>5 class and 7 source code structural measures (2 were found correlated with effort)</td>
<td>effort (project-wide) and source code complexity</td>
<td>oo late design phase</td>
<td>no</td>
<td>pearson’s r, linear exp LS</td>
<td>7 proj, small sw comp.</td>
</tr>
</tbody>
</table>
TABLE 1:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretical</th>
<th>Empirical</th>
<th>Environmental</th>
</tr>
</thead>
<tbody>
<tr>
<td>...morasca [184]</td>
<td>3 size, 1 length, 3 structural complexity, 2 coupling (I could list the measures)</td>
<td>concurrent software specifications in petri nets</td>
<td>property based (bridand)</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...morasca russo [185]</td>
<td>FF, LOC, num of modules (NOM), num of employees, relative experience on the application, start date of activities.</td>
<td>productivity of FF, LOC, NOM, software development and maintenance</td>
<td>no</td>
<td>univariate logistic regression</td>
<td>Ital publ admin envir</td>
<td></td>
</tr>
<tr>
<td>...moser et al. [186]</td>
<td>Dome System Meter Function Points and Function points</td>
<td>effort (project-wide)</td>
<td>information systems</td>
<td>no</td>
<td>univariate LS</td>
<td>37 projects</td>
</tr>
<tr>
<td>...moses et al. [187]</td>
<td>information flow architectural and detailed design (IFAD, IFDD), fanin, fanout</td>
<td>cohesion</td>
<td>modules at architectural and detailed design levels</td>
<td>properly based of briand also empirical systems</td>
<td>binary logreg</td>
<td>academ exp.</td>
</tr>
<tr>
<td>...nagappan [188]</td>
<td>STREW suite (num of test cases/LOC, num of test cases/num of req, testlines of code/LOC, num of asserts/LOC, code coverage)</td>
<td>software reliability, quality of testing effort</td>
<td>oo language</td>
<td>no (but planned)</td>
<td>ols, 13 senior student programs (1 study)</td>
<td></td>
</tr>
<tr>
<td>...nesi querci [189]</td>
<td>complexity and size (method level, class level, system level metrics)</td>
<td>class effort</td>
<td>oo C++ systems</td>
<td>NO but well def.</td>
<td>multilinear regr</td>
<td>3 sw industrial</td>
</tr>
<tr>
<td>...orme et al. [190]</td>
<td>num of external classes, reference to external classes, reference includes</td>
<td>coupling (subject's opinion)</td>
<td>ontology based system</td>
<td>meas. theory (kitch et al)</td>
<td>Pearson</td>
<td>commercial software</td>
</tr>
<tr>
<td>...phalp et al. [191]</td>
<td>coupling within RAD (role coupling factor, and system coupling factor) maybe I should skip this paper because is badly written</td>
<td>project characteristics (development process, project size, static business process models (role activity diagrams)</td>
<td>property based [104]</td>
<td>correlation (maybe spearman because of ranking)</td>
<td>2 case studies</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1: Referential Measures Attribute Entity Theoretical Empirical Environment

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretical</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>---piat-tini et al. [193]</td>
<td>structural complexity (num of: attributes, derived, composite and multivalued attributes, 1-ary, 2-ary, M-ary, N-ary relationships, IS_A, reflexive, and redundant relationships)</td>
<td>subject rate of maintainability (simplicity, analysability, understandability, modifiability, stability and testability)</td>
<td>EKD</td>
<td>brand property based</td>
<td>induction of fuzzy rules</td>
<td>academ exp</td>
</tr>
<tr>
<td>---piat-tini et al. [194]</td>
<td>num of attributes, depth of the referential tree, referential degree combination of DRT and RD</td>
<td>subject rate of maintainability (analysability, changeability, stability, compliance and testability)</td>
<td>relational databases</td>
<td>zuse framework</td>
<td>F-statistic</td>
<td>academ exp</td>
</tr>
<tr>
<td>---pons et al. [195]</td>
<td>polymorph metric based on hierarchies, classes, methods, core hierarchies, width, and children</td>
<td>polymorphism</td>
<td>conceptual model of a system in UML</td>
<td>kitch and zuse</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>---ragunathan et al. [196]</td>
<td>aggressive promotion, analysis-based development, defensive management, future-oriented development, proactive management, conservative management of information systems</td>
<td>IS performance effectiveness (measured on a five points likert scale)</td>
<td>information management strategy</td>
<td>content, unidimensionality, reliability, discriminant validity with guidelines from IS area</td>
<td>t-test</td>
<td>231 IS executives, industrial</td>
</tr>
<tr>
<td>---rajaraman [197]</td>
<td>class inheritance and non inheritance related coupling, class coupling, average method coupling</td>
<td>perceived complexity</td>
<td>C++ software</td>
<td>no</td>
<td>Pearson’s r</td>
<td>5</td>
</tr>
<tr>
<td>---rauterberg [198] [199]</td>
<td>functional feedback, interactive directness, application flexibility, dialog flexibility</td>
<td>usability (task solving time, target discrepancy)</td>
<td>user interface characteristics</td>
<td>no</td>
<td>compar. of descriptive data, ANOVA</td>
<td>laboraatory exp</td>
</tr>
<tr>
<td>---rey-noso, genero et al. [200]</td>
<td>a set of measure for uml/ocl language</td>
<td>understandability, modifiability</td>
<td>object constraint language (OCL) expressions</td>
<td>property based (brand)</td>
<td>spearman, tau kendall</td>
<td>3 experiments academic</td>
</tr>
<tr>
<td>---rossifernandez [201]</td>
<td>8 structural, 6 behavioural</td>
<td>---</td>
<td>distributed applications</td>
<td>distance framework</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
TABLE 1:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
<th>Attribute</th>
<th>Entity</th>
<th>Theoretic al</th>
<th>Empirical</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>---ruhe et al. [202]</td>
<td>size measures (function points, web objects)</td>
<td>subjective and measured estimation of development effort</td>
<td>web applications</td>
<td>no</td>
<td>t-test</td>
<td>small dev company, 12 appl</td>
</tr>
<tr>
<td>---soward et al. [203]</td>
<td>perceived information scent (age, gender, internet experience, exp with retailer online, exp with retailer stores, preferred navigation method)</td>
<td>system usability (task completion, task time, user perception of easy use, target item in expected place)</td>
<td>web or internet information retrieval systems</td>
<td>no</td>
<td>kendall's tau</td>
<td>e-commerce application, acad study</td>
</tr>
<tr>
<td>---sercano et al. [204]</td>
<td>number of fact and dimensional tables</td>
<td>understandability</td>
<td>multi-dimensional data models</td>
<td>performed in [119] with zuse framw</td>
<td>f statistic</td>
<td>academi exp</td>
</tr>
<tr>
<td>---sharma et al. [205]</td>
<td>LCOM, mult (number of classes that uses multiple inheritance), number of polymorphic dispatches (NOPD)</td>
<td>complexity</td>
<td>oo systems</td>
<td>weyuker's property 9</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>---stenrud et al. [206]</td>
<td>MMRE</td>
<td>project size</td>
<td>prediction systems accuracy</td>
<td>no</td>
<td>t-test, Mann - Whitney, scatter-Plots</td>
<td>5 large datasets</td>
</tr>
<tr>
<td>---Subraman-yam and Krishnan [207]</td>
<td>complexity metrics (subset of CK: WMC, CBO, DIT)</td>
<td>fault proneness</td>
<td>oo applications</td>
<td>no</td>
<td>weighted linear regres</td>
<td>industrial data</td>
</tr>
<tr>
<td>---tang et al. [208]</td>
<td>chidamber kemerer set, inheritance coupling, coupling between methods, num of object memory allocation, average method complexity</td>
<td>fault proneness</td>
<td>oo classes?</td>
<td>no</td>
<td>logistic regression</td>
<td>industrial sw</td>
</tr>
<tr>
<td>---wood wdmr99 [209]</td>
<td>flat vs deep inheritance structure</td>
<td>maintainability (time for maintenance task)</td>
<td>oo software</td>
<td>no</td>
<td>Wilcoxon</td>
<td>3 academ exp</td>
</tr>
<tr>
<td>---wilkie wk98 [210]</td>
<td>C&amp;K set; various interpretations for WMC (McCabe, Halstead)</td>
<td>effort for field fault fixes; effort for functional enhancements</td>
<td>c++ class</td>
<td>no</td>
<td>uni/multi linear ls</td>
<td>conferencing system, 114 C++ cls.</td>
</tr>
<tr>
<td>---wilkie wk00 [211]</td>
<td>CBO, #public functions per class, #functions per class</td>
<td># and proportion ripple changes a class participates in, changes in a class</td>
<td>c++ class</td>
<td>no</td>
<td>Kruskal Wallis, Kendall's tau</td>
<td>---</td>
</tr>
</tbody>
</table>
The measures in the table (column measures) are not all validated. That is, the authors have applied methods to validate the measures theoretically or empirically without success [refs of the papers with such cases]. Furthermore, there is a lack in the literature of the relationship between theoretically and empirically validated measures. Two cases can happen: 1. a measure is theoretically valid but not empirically; 2. a measure is empirically valid but not theoretically (the cases where the measures are valid/not valid both theoretically and empirically are banal, do not need any further discussion). What do we do with these measures? According to Briand (and maybe others as well) the measures have to be validated in both senses and we should throw away the non validated measures. However, the practitioners (probably) do not care whether a measure is theoretically validated, while importance is given to the empirical validation which demonstrate that the measure is useful. There is not so much discussion in the literature [mention those papers which have some discussion ie gursaran [153] and briand 95 [19]], and questions like “does the theoretical validation affect the empirical?” and vice versa are rarely discussed.Discussion

Except for object oriented measures, it is not possible to create a body of knowledge for software measures in many areas of software engineering. I summarise here the main reasons:

- There are many isolated studies. Several measures have been empirically validated only one time which is not sufficient to generalise the results. The most common entities measured are object oriented design and source code and conceptual data models. More entities (requirements, processes, resources) and more attributes need to be studied (all the ilities...). Few replications have been performed to confirm the results of the validation.

- Many of the measures proposed lack theoretical validation. Furthermore, a measure can be theoretically validated with one method but not validated with another method. Which method is the best?

- Few studies discuss how the theoretical validation affects the empirical one and vice versa. Is it useful to theoretically validate the measures? How does the theoretical validation affect the empirical one (and vice versa)?
• Studies are often context dependent, therefore they need to be replicated in different environments where large data sets and larger systems are available for investigations. Alternatively, the measures have to be used locally.

• Comparison of prediction models with expert opinion is rare. According to Briand [ref, page 7], “an interesting question that, to our knowledge, has not been investigated in depth to date is whether structural measures can perform as well as or better than experts in predicting quality attributes such as fault proneness”. The only investigation of this kind is done in [apsec, ...]....

I want to stress that few validations have been performed on requirements measures except [internal report, apsec, joiast, ambriola,]. As already pointed out by briand “the use of predictive models based on early artifacts and their capability to predict quality of the final system still remains to be investigated” [ref].

Further work is necessary to validate measures theoretically and empirically and to study the relationships between the two kinds of validations. (Empirically) Validated measures are needed for managers and practitioners in order to take better decisions which is the goal that any measurement program must pursue in order to be useful.

empirical validation and regression: 2 papers selected
empirical validation and statistic anal.: 5 selected
empirical validation and software quality:

why do I restrict the search to software engineering? well it would be nice to check how validation is done in other areas too, but there are many areas where measures are used and validated (not only in software engineering). I had to restrict the search in some way. At least in this way the table is complete in software engineering (but, I have to check some validations done in other areas!). Regression analysis and prediction models are very much used in artificial intelligence, I should check some methods, like baesyan networks, maybe even Hidden markov models to see if some methods are applicable in my case.

there is one paper of arisholm emp soft eng vol 6 n 3, to be checked

it is not sure the table is complete, some researchers do not use our chosen keywords (for instance they could use prediction models, or similar).

note that one validation of function points says that there are other validation of function points which I have not referred.

note that gursaran make an empirical validation to demonstrate that the represntation condition is satisfied.

note that I have considered mainly the empirical validations, ther are a lot of papers on prediction models that could also be considered as empirical validations. Few of them are in the table but not many.

note the measures in bold are validated
note, new problems: if I want to say what measures are validated and which are not it is not so simple, because sometimes there are several dep attributes and and there are several possible combinations. sometimes a combination of metrics is validated but not the single ones, sometimes they are validated only in special cases.

note: when I say no in theoretical validation, I mean that the authors do not mention anything, however, sometimes the measures have been validated somewhere else anyway (like in some papers of briand)
Discussion of papers

In this chapter I present a summary of the papers included in the thesis. For some papers, there are also discussion and future work. Furthermore, I will classify each paper giving the research approach and research method used (see glass information and software technology 44 2002)

5.1 Paper 1


A practical way to help companies to manage their requirements is through software measurement. Therefore the purpose of this paper is to define a wide set of software measures that can help to satisfy the goals of the Requirements Management (RM) Key Process Area (KPA) of the Capability Maturity Model (CMM), and to ease the adoption of the KPA.

In this paper I describe briefly the RM KPA of the capability maturity model and the goal question metrics. After that, I apply the Goal/Question/Metric (GQM) to the Requirements Management KPA. The two predefined goals of the RM KPA are re-defined using the GQM template for the goals definition. The result of the GQM application is a set of 15 questions and 33 measures for the first goal, and a set of 7 questions and 12 measures for the second goal. A total of 20 questions and 38 measures is presented. The set of measures defined is easy to understand and to use, it can be used as a pick list, the measures are general (they are not context dependent) and comprehensive.
The paper also provides a guidance for people trying to implement the Requirements Management KPA and inform with lessons learned and practical results. The metrics obtained will help immature companies to satisfy the goals of the Requirements Management KPA.

This work can be extended in many ways for instance by creating subgoals and grouping the measures defined according to the subgoals. Furthermore, each measure could be associated to the entity and the attributes they are supposed to measure. The measures need to be validated both theoretically and empirically in order to create a robust set of measures that satisfy the basic measurement theory principles (see chapter 2). This is partially done in papers 2 and 3. Among the measures defined, there is the "size of requirements". In my subsequent papers, this measure is decomposed in number of lines, number of words, number of actors and number of use cases. These measures are empirically validated in papers 5, 6, 7, 8.

The measures defined are many, therefore companies are suggested to tailor the wide set of measures to their own needs, depending on their maturity. Companies are also suggested to set priorities among the measures and thresholds for each measure, and this can help decision makers in their activities.

5.2 Paper 2


In this paper I perform a theoretical validation of the measure "number of requirements" which can help to manage the requirements and to know their volatility. The reason to theoretically validate measures is because it is necessary to show that a measure is associated to the attribute it purports to measure [ref briand el el man]. I apply two different procedures to validate the measure. This will show that there are conflicts in the principles of measurement theory available in the metrics literature. This example of theoretical validation shows the difficulties in adhering to a strict measurement approach. Infact, the measure is still not validated because of the impossibility to prove that the representation condition is satisfied both theoretically and empirically. Several questions are arisen and discussed and they confirm the immaturity of the measures validation field. My goal was to perform a theoretical validation of the measure independently of the context. This is not possible, the theoretical validation has to be done while performing an empirical study. The theoretical validation of the measures is context dependent and therefore has to be re-done everytime the measures are needed in a new context (comment: this make the construct validity in all the empirical validations not really valid anymore). This lead me and my co-author to perform a theoretical validation of ten of the measures presented in paper1. We could not validate all the measures because the project analysed was small and we could not collect data for all the measures in paper 1. The validation is presented in paper 3 and it is performed in an academic environment.
5.3 Paper 3


The goal of this paper is twofold. First we theoretically validate a subset of ten requirements measures from the set of 38 measures defined in paper 1. After that we describe a case study where we use the validated measures to estimate the cost of changes to requirements. The paper starts with a description of basic measurement theory, including advantages and disadvantages of software measurement, and theoretical and empirical validation approaches. We describe the entities, which are organised in a hierarchical structure, and the internal attributes related to the measures to be validated. The measures are then validated theoretically by applying two validation procedures to the ten measures under analysis. Measures need also to be validated empirically, by demonstrating that the measures are connected to some important external attribute. Therefore we performed an academic case study whose goal was to demonstrate that cost estimations based on historical data are better than intuitive cost estimations i.e. we want to make cost predictions based on data from our measures. Unfortunately, we were not able to collect enough data to draw significant conclusions. The problems with the study were due to low control of the student’s projects and also for missing mandatory requirement for collecting data.

This work is unique because, to our knowledge, there are no requirements management measures theoretically validated. The focus of the scientific community is more on product measures, especially for source code (as we have seen in chapter 4). The contribution of the case study is on the methodological side. It is a good experience which can teach what issues have to be considered when performing an academic case study.

This work can be extended in many ways for instance by applying other theoretical validation approaches [briand et al., zuse, and the distance metric poels and deehene]. In order to validate the measures left from paper one it is necessary to perform a large scale case study, larger than those described in papers 5, 6, 7, 8. Definitions of change, requirement, size of requirement, and size of change need to be included together with a more detailed description of the verification of the representation condition. Furthermore, the study could be replicated by tightening the goals with educational goals.

5.4 Paper 4


In this paper, we describe partial results of an academic case study made in autumn 2003 and compare it with a previous one, performed in autumn 2002. The first study was executed with the purpose to investigate whether cost estimation of changes to requirements per-
formed using historical data are better than intuitive cost estimations. This study is the same described in paper 3. The purpose of the second study was to investigate whether cost estimation of changes to requirements based on detailed impact analysis are better than intuitive cost estimations. Another goal of this paper was to compare the case studies and show the methodological improvements accomplished by using the lessons learned from the first study. The final results of case study two and its comparison with study one are listed below, together with some lessons learned.

Final results
Data was collected from week 39 to week 03 of 2003-2004. Four teams submitted all data, not regularly, only team 5 submitted data on a regular base. The other teams missed the weekly deadlines several times. The data of team 4 is incomplete. The data obtained by team 3 seem not reliable because one change took 75 hours to implement. This can be due to the Hawthorn effect. Teams 3, 4, and 5 have performed estimations following an impact analysis checklist while teams 1 and 2 have performed intuitive estimations. Considering a comparison between teams 1 and 2 (non controlled teams) and team 5 (controlled team), there is no difference in the estimations. Therefore we can conclude that the estimations based on a detailed impact analysis checklist are not better than intuitive estimations. However, the data collected were incomplete and unprecise therefore we need to make replications of this study to be able to draw general conclusions.

Lessons learned in study two
As pointed out earlier, the study did not succeed as expected for two main reasons. First: among the five teams performing a project in the OOSD course, four teams collected all data. Furthermore, the data from team 3 is not reliable. Secondly: the requirements were not as volatile as expected. According to the original plan, the controlled teams (teams 3, 4, 5) should do cost estimation of changes to requirements based on detailed impact analysis. However we have learned some lessons, which are listed below.

- It is important to keep track of change requests status
- The amount of data to be collected should be kept small

<table>
<thead>
<tr>
<th>Measures</th>
<th>Study one</th>
<th>Study two</th>
</tr>
</thead>
<tbody>
<tr>
<td># Requirements</td>
<td>Team a 31</td>
<td>Team b 12</td>
</tr>
<tr>
<td></td>
<td>Team 1 30</td>
<td>Team 2 9</td>
</tr>
<tr>
<td></td>
<td>Team 3 17</td>
<td>Team 4 28</td>
</tr>
<tr>
<td></td>
<td>Team 5 25</td>
<td></td>
</tr>
<tr>
<td># Changes</td>
<td>28</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Estimated Time per change</td>
<td>23.3</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>95 or 165?</td>
<td>135</td>
</tr>
<tr>
<td></td>
<td>4800</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>Actual Time per change</td>
<td>27.3</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>68</td>
<td>195</td>
</tr>
<tr>
<td></td>
<td>4500</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>Error</td>
<td>-15%</td>
<td>-60%</td>
</tr>
<tr>
<td></td>
<td>+40%</td>
<td>-30%</td>
</tr>
<tr>
<td></td>
<td>+7%</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+40%</td>
</tr>
</tbody>
</table>
- Requirements management is not an area to be investigated in academic course context.
- In case of empirical studies performed in a course context, collection of data must be a requirement to pass the exam.

As a general conclusion of the two studies, we can say that students courses are not suitable for studies with changing requirements, the courses are too short and the projects are small and stable. Furthermore, we cannot draw any conclusions about the measures and their usefulness. Some of these measures are therefore studied furtherly in papers 5, 6, 7, 8.

5.5 Paper 5


Requirements volatility (RV) is a very complex phenomenon that is still not well understood and hence known measures of RV are rare to find (neither in literature nor in industry). Hence, this paper tackles an important issue worthy of investigation especially empirical investigation. This paper reports on a case study conducted to investigate different measures of requirements volatility on a project. The goals of the study were: 1) to prove that a set of requirements size measures are associated with the volatility of use case models (UCM); 2) to investigate the correlation between subjective and objective volatility. The study analyses whether the size of a requirement and the size of a requirements change provide information about a requirement’s propensity to change in the future, thus giving some indication of volatility. Its focus is actually on whether this correlates with developers’ own perceptions about volatility. The measures were obtained from our previous work (paper 1 and paper 3) choosing in particular the measures related to volatility. Measurement data was collected in retrospect for all use case models of the software project. In addition, we determined subjective volatility by interviewing 10 stakeholders from a company in Sweden. They completed a questionnaire in which they were asked about the perceived volatility of 14 use case models. The data analysis showed a high correlation between our measures of size of UCM and total number of changes, indicating that the measures of size of UCMs are good indicators of requirements volatility. The study found no correlation between perceptions and actuality. These results suggest that project managers at this company should measure their projects because of the risk to take wrong decisions based on their own and the developer’s perceptions. The larger the UCM, the more potential there would be for errors. Therefore, the results serve to re-emphasise some fundamental software engineering principles regarding the need for modularity and cohesion in order to manage complexity and localise change.

We analysed a completed real project in retrospect, therefore this study has the advantage of not affecting the outcome of the project. The project analysed and the number of people interviewed is small and in order to generalise the results we need to replicate this retrospec-
tive case study which will be done in paper 8. An interesting extension of this study would be to confront the stakeholders with the real change data after they did their estimates and let them comment about the differences.

The changes to requirements and its classification in size needs to be investigated further. The way we defined change may be different from the subjects’ views. In fact, it has been observed in practice that people have different views on what constitutes requirements change. It is often confused with bug report or defect fixing. Furthermore, the classification of changes in minor, moderate, and major is subjective and was not evaluated. It would be interesting also to investigate how to handle dependencies between the changes in different classifications e.g. what if a minor change impacts on a major change elsewhere? Counting the total number of changes (and mixing the sizes of changes) could be misleading because the correction of a typo is counted the same as replacing an actor. Furthermore, the way we classified the size of change might mask the true size, for instance when a spelling mistake is a change that causes the meaning of the requirement to change. The construction of prediction models (papers 6 and 7), is based on the results of this study. However, we decided to change the measurement rules and to count the number of words changed. In this way the data collection becomes objective and can be automated.

5.6 Paper 6


In this paper we describe a correlational study with the goal of empirically validating five measures of requirements size as predictors for the number of requirements changes. Although, the number of changes does not measure volatility directly, it is an important basic measure that can be used to easily compute other measures, like change density or change frequency. The study is explorative, i.e. we analyse the data collected for our measures to find out the best predictor of number of changes. To our knowledge, no empirical validation of requirements change measures as predictors has been performed in an industrial setting. We built seven prediction models using data collected for a medium-size software project developed at BAE Systems Hägglunds AB, Sweden. The context of the study and the data collection are described in paper 5. The accuracy of five models was evaluated by applying them on a set of data collected for a second project at the same company. We performed a cross systems validation. Our best model showed a pred(0.25)=0.5, which is better than the accuracy of common effort prediction models like for example COCOMO. The results show that the best predictors of number of changes are the length measures: number of lines and words. Other predictors of complexity and functionality were found less accurate. The prediction models built, can help project managers to estimate the volatility of requirements and minimize the risks caused by volatile requirements, like schedule and costs overruns. Although our models are likely to have only local validity, the general method for constructing the predic-
tion models could be applied in any software development company. In an earlier study, we showed that decisions solely based on developer perception are unreliable. Predictions models, like the one presented here can help to mitigate that risk.

In this study we choose number of changes as a dependent variable but it is not a measure of volatility. Volatility is a complex concept and depends on many factors. In our work we demonstrate that size is one of these factors. In paper 7 we describe another correlational study analysing the same data used in this correlational study. The difference is the dependent variable volatility which is defined as the sum of change densities in time.

5.7 Paper 7

In this paper, we present a correlational study with the goal of predicting requirements volatility for a medium size software project. Based on the data collected from two industrial software projects for four measures of size of requirements (number of actors, use cases, words, and lines), we have built and evaluated prediction models for requirements volatility. Performing a cross systems validation, the best model showed a MMRE = 0.25, which can be considered reliable. Although our models are likely to have only local validity, the general method for constructing the prediction models could be applied in any software development company.

5.8 Paper 8 replication
disadvantages of replication:
Conclusions

6.1 Summary

Definition of 38 requirements management measures
Theoretical validation of ten measures
Empirical validation:
2 academic case studies
1 industrial case study
1 correlational study

list of all the ways I have measured volatility
list the results of

6.2 Contributions

38 measures defined
10 measures theoretically validated
4 measures proven to be good assessors
2 length measures are the best predictors
The developers' perceptions were not good estimators of volatility of past projects
6.3 Limitations, challenges and open issues

maybe the results need to be presented in a way so that the customers and practitioners can understand and be careful to not increase the gap between the researchers world and the real world. [pfleeger ieee software 97]

   Progress in solving the stated problem
   more studies are needed
   Method of evaluation
   open questions
   measuring process or product?
   Atomic requirements?
   Which theoretical validation is the best?
   Is it useful?
   has some meaning to talk about the size of requirements? and size of change?
   Are your measures independent of the context? measurement cannot be defined independent of context several factors come into play(IEEE software april 99 forse 00)
   Cultural barriers: people don’t like to measure, get bored especially when they don’t understand why they measure.

this work can be improved in the following way:

6.4 Future work

Closing remarks
   (have I reached my goals?)
   is it new?
   is it better than other things done earlier?

   It would be interesting to examine the relationship between the volatility of individual requirements and the volatility of the requirements body as a whole, both actual and perceived.

   project similarity

from apsec:
subjective volatility was not good estimator of number of changes
size measures were good estimators of total number of changes.
(no comparison among the structural measures and the subjective measures)

from technical report
the structural measures are good predictors of total number of changes (on both projects)

from just submitted ase2007
the structural measures are not all good predictors of volatility
num of lines is a good predictor of volatility (on both projects)
from the replication
the subjective measures are not good estimators of total num. of changes and of volatility
the structural measures are not good estimators of volatility
the structural measures are good estimators of total num of changes
maybe?? the structural measures are better estimators then the subjective estimators.

partial conclusions:
number of lines and words are good estimators of number of changes, in all the studies.
We cannot say anything about Number of use cases, actors and scenarios, they were not
good estimators of number of changes in all studies.
Number of lines is a good predictor of volatility on both projects according to the corre-
lational study
number of lines is not a good estimator of volatility according to the case study. This result
is in contradiction with the correlational study. The fact is that in the correaltional study we
used a prediction model (ax+b) and not the simple number of lines
The other measures are not good predictors of volatility (in all studies)
The subjective volatility was not good

final conlcusion:
whatever measure of volatility we choose, the subjective volatility is not good predictor
what is the best way to measure volatility of single requirements?
the structural measures seem good predictors of volatility (in whatever way we measure it)
References

7.1 References from chapter 1

7.2 References from chapter 2


[30] Futrell shefer shefer?


[36] juristo moreno


[41] Kitchenham, B.A. Pfleeger, S.L. fenton, reply to comments on Towards a Framework for Software Measurement Validation IEEE 23 3


[48] Natt och Dag thesis


[50] Olafsson v-gqm metrics 2001


[52] Perry votta 2000


[54] Pfleeger ieee computer oct 99


[56] Pfleeger ieee software 2005

[57] Poels and dedene

[58] Pour, Gilda martin Griss and Michael Lutz The push to make software engineering respectable. IEEE computer may 2000.


[60] Shaw mary ieee software november 90

[61] Shaw M The coming-of-age of software architecture research ICSE 2001 Shaw 2001, (not read yet

[62] Shneiderwind, ieee computer 2002


[66] Tichy Walter F. ieee computer 98

[68] Van solingen, G.


[71] Xia, frank, joist 1999


[73] Zelkowitz MV, D Wallace Experimental validation in software engineering Information and Software Technology 39 (1997) 735-743

[74] Zelkowitz MV, D Wallace (ieee computer may 98) experimental models for validating technology

[75] Zelkowitz MV, Sew proceedings 98

[76] Zendler 2001


7.3 References chapter 4


[82] VS Alagar, Q Li, OS Ormandjieva Assessment of maintainability in object-oriented software, 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39) p. 0194 IEEE Computer Society Washington, DC, USA


[89]---B. Bahli, Suzanne Rivard, A Validation of Measures Associated with the Risk Factors in Information Technology Outsourcing 36th Annual Hawaii International Conference on System Sciences (HICSS’03) - Track 8 January 06 - 09, 2003 Big Island, Hawaii


[93]---B Baudry, Y Le Traon, JM Jezequel, Robustness and Diagnosability of Designed by Contracts OO Systems ,Proc. 7th International Software Metrics Symposium, p. 272 IEEE COMPUTER SOCIETY


[100]


[102]---Briand, L.C., J Daly, V Porter, J Wüest, Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems, /isse/1998

[103]---L.C. Briand, JW Daly, J Wüest - A Unified Framework for Cohesion Measurement in Object-Oriented Systems Empirical Software Engineering Volume 3, Number 1 Date: March 1998 , Pages: 65 - 117


[105]---L.C. Briand, J Daly, V Porter, J Wüest A comprehensive empirical validation of design measures for object-oriented systems - Software Metrics Symposium, 20-21 Nov 1998 page(s): 246-257


[117]---Coral Calero, Mario Piattini, Marcela Genero A Case Study with Relational Database Metrics ACS/IEEE International Conference on Computer Systems and Applications (AICCSA’01) June 25 - 29, 2001 Beirut, Lebanon


[126]—SR Chidamber, CF Kemerer Towards a metrics suite for object oriented design Conference on Object Oriented Programming Systems Languages and Applications Phoenix, Arizona, United States 211, 1991 ACM Press New York, NY, USA

[127]—Costagliola Filomena Ferrucci Genoveffa Tortora, Giuliana Vitiello Class Point: An Approach for the Size Estimation of Object-Oriented Systems, IEEE TSE, January 2005 (Vol. 31, No. 1) pp. 52-74


[129]—Melis Dagpinar, Jens H. Jahnke Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison 10th Working Conference on Reverse Engineering November 13 - 17, 2003 Victoria, B.C., Canada

[130]—A De Lucia, E Pomella, S Stefanucci - Assessing effort estimation models for corrective maintenance through empirical studies, Information and Software Technology 47:3–15, 2005

[131]—P Devanbu, Sakke Karstu, Walcelio Melo and William Thomas, Analytical and Empirical Evaluation of Software Reuse Metrics, 18th International Conference on Software Engineering (ICSE'96) p. 189/ice/1996/


[141]---F. Fioravanti, A Metric Framework for the Assessment of Object-Oriented systems IEEE International Conference on Software Maintenance (ICSM’01) November 07 - 09, 2001 Florence, Italy


[143]genero defining and validating metrics for conceptual models, phd thesis, 2002


[146]---Genero, Poels Piattini, Defining and validating measures for conceptual data model quality, Lecture Notes In Computer Science; Vol. 2348 Proceedings of the 14th International Conference on Advanced Information Systems Engineering LNCS2348


[151]---M. Genero, M. Piattini, E.Manso, G. Cantone, Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics , metrics 2003


[156]---GA Hall, W Tao, JC Munson Measurement and Validation of Module Coupling Attributes, Software Quality Journal, 13, 281-296, 2005 Springer Science


[162]---T.e. Hastings A.s.m. Sajeev, A Vector-Based Approach to Software Size Measurement and Effort Estimation, IEEE TSE, April 2001 (Vol. 27, No. 4) pp. 337-350, theor empirical valid of approach


[166]---MY Ivory, UC Berkeley, RR nSinha, MA Hearst Empirically Validated Web Page Design Metrics Proceedings of the SIGCHI conference on Human factors in computing
systems, Seattle, Washington, United States Pages: 53 - 60 2001 ACM Press New York, NY, USA


[174]loconsole internal report

[175]loconsole apsec 2005

[176]loconsole jss 2006


[178]----M. E. Manso, Marcela Genero, Mario Piattini No-redundant Metrics for UML Class Diagram Structural Complexity, CAiSE 2003, Lecture Notes in Computer Science Publisher Springer Berlin/Heidelberg Volume 2681/2003, 127-142


[182]—D Miranda, M Genero, M Piattini Empirical Validation of Metrics for UML State-chart Diagrams, Fifth International Conference on Enterprise Information Systems (ICEIS 03), Vol 1., 87-95.


[184]—S Morasca, Measuring Attributes of Concurrent Software Specifications in Petri Nets, Sixth IEEE International Symposium on Software Metrics, November 04 - 06, 1999 Boca Raton, Florida


[192]—M. Piattini, M. Genero, C. Calero, Data Model Metrics, Handbook of Software Engineering & Knowledge Eng, 2001 (not for table)


[201]---P Rossi, George Fernandez Definition and Validation of Design Metrics for Distributed Applications Ninth International Software Metrics Symposium (METRICS’03) p. 124


[208]---Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen, An Empirical Study on Object-Oriented Metrics, Sixth International Software Metrics Symposium (METRICS’99) p. 242 metrics/1999


Abstract

The purpose of this paper is to provide software measurements for implementation of the goals of the Requirements Management Key Process Area (KPA) of the Capability Maturity Model (CMM), and to ease the adoption of the KPA. The paper also provides practical guidance for people trying to implement the Requirements Management KPA. The CMM, developed by the Software Engineering Institute (SEI) is not well supported by software measurement and it is somewhat complex. An application of the Goal/Question/Metric (GQM) paradigm to the Requirements Management KPA is therefore presented. The metrics obtained will help immature companies to satisfy the goals of the Requirements Management KPA.
1.1 Introduction

Software processes are considered to be the main area for quality improvement. There are two main streams within Software Process Improvement (SPI) [17]. One is based on assessments of organisations’ capability, e.g. Capability Maturity Model for Software (SW-CMM) [12], Software Process Improvement Capability dEtermination (SPICE) [6], BOOTSTRAP [11], and the ISO9000 family. The other is based on measurements of software practices within an organisation, e.g. Goal/Question/Metric (GQM) [1], Quality Improvement Paradigm (QIP) [2], and Application of Metrics in Industry (AMI) [14]. These approaches complement each other because software measurement is inherent to the concept of improvement, but they are seldom applied together [17]. The assessment-based approaches should always include measurements, because it is necessary to estimate the state of the software process before action is taken to improve it and compare it with the state thereafter. The SW-CMM developed by the Software Engineering Institute is intended to help software organisations to improve the maturity of their software processes, but is weakly supported by a measurement-based approach. Therefore this paper proposes a set of measures for the Requirements Management Key Process Area (KPA) of the SW-CMM. This work has been done with the aim of joining the assessment and measurement based methodologies mentioned above. The underlying assumption in the paper is that it is easier to focus the measurement and improvement activities on a specific process area, rather than trying to measure and improve all the process areas at once. This is especially true for small-medium size enterprises, which do not have enough resources to train people on complex frameworks like the SW-CMM. The Requirements Management KPA has been chosen because it is important to control the continuing definition of requirements as they change throughout the software life cycle. Such control over the requirements helps in anticipating and responding to requests of change [16]. The aim has been addressed by analysing the Requirements Management KPA of the SW-CMM and its key practices [13], and applying the GQM paradigm to it.

The first contribution of this paper is to give a general and comprehensive set of software measures for the implementation of the goals of the Requirements Management KPA within the SW-CMM (see [7] for a definition of measure and measurement). This set of measures constitutes a "pick list" that can be tailored to the specific enterprise, offering small to medium size enterprises the freedom to choose suitable subsets of software measures. Another contribution of this paper is a simple method of improving the Requirements Management activity and the presentation of basic software measures, which are easy to understand and to use. The paper presents a practical approach and a practical guidance for people trying to fulfil the goals of the KPA. The method can also be used to evaluate the state of the Requirements Management activity.

At the time of writing, the measures are being tested at Ericsson Erisoft AB in Umeå, Sweden, which is a medium size company. The results of this empirical study will help the author to demonstrate that a joint approach is more complete than an assessment or meas-
urement based approach. The measures obtained can be used in quantifying the amount of changes to requirements and to predict the cost for a change, helping to control requirements and changes to requirements.

The measures will be grouped by maturity level. For instance an immature enterprise can start measuring the total number of requirements and the number of changes to requirements throughout the software life cycle. This would provide better control and visibility into the Requirements Management activity, improving the software process a small step towards the goal of having a repeatable process. For each measure proposed, improvement actions will be suggested to help decision makers in their job. After these actions are taken, an enterprise can follow the approach described in "Application of Metrics in Industry" [14], which suggests to cycle four steps (analyse, act, metrificate, and improve) to continue with the improvement of the KPA.

The remainder of this paper is organised as follows: the section 1.1 describes related research in software measurement for the SW-CMM, sections 2 and 3 present an overview of the CMM and the GQM respectively, the application of the GQM to the CMM is described in section 4, section 5 contains initial results of the experiment started in the before-mentioned company, and concluding remarks and future directions are presented in section 6.

Related Work
Several studies on software measurement for the SW-CMM have been done prior to this work. One of the most relevant has been done by Baumert and McWhinney [3]. Their report provides a set of indicators (composite measures) that are compatible with the measurement practices of the SW-CMM. The indicators are categorised according to the quality attributes they fulfil and not by KPAs. Raynus [15] confirms the connection between software measurement and the SW-CMM. His work examines relationships between measurable process quantities, and reviews the SW-CMM demonstrating that software measurement can be used to improve the behaviour of a software development organisation. His book represents a quantitative approach to software management and SPI. He suggests some metrics for the SW-CMM and their use. In goal oriented software improvement, the measurements must be focused on specific goals, for example specific process areas, while the authors mentioned above, suggest indicators across all the maturity levels. Therefore, this paper presents a comprehensive collection of measures that are focused on a specific KPA namely the Requirements Management KPA of the SW-CMM.

1.2 The Requirements management KPA of the capability maturity model
As shown in figure 1, the CMM is composed of 5 distinct levels (Initial, Repeatable, Defined, Managed, Optimising) [12].
Each CMM level, except the initial level, has several Key Process Areas (KPA). [12]. One level 2 KPA is Requirements Management. "The purpose of Requirements Management is to establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project" as defined in [13]. This means that the requirements of a software project should be complete, documented, unambiguous, controlled, etc., in order to design a software product, which satisfies the customer's needs. Very often, requirements change through the software development life cycle but the control of the change requests is poor. The activity of "Requirements Management" is focused on the control of the requirements gathering, establishing an agreement between the
customer and the software team on the requirements, checking, reviewing and managing the changes on requirements. This activity is the process of ensuring that a software product produced from a set of requirements, will meet those requirements.

1.3 The Goal/Question/Metric paradigm

The Goal/Question/Metric (GQM) paradigm is a method for helping an organisation to focus the measurement program on their goals. It states that an organisation should have specific goals in mind before data are collected [1]. GQM does not specify concrete goals. It is rather a structure for defining goals and refining them into a set of quantifiable questions. These questions imply a specific set of metrics and data to be collected in order to achieve these goals. The GQM paradigm consists of three steps:

Purpose:
- Analyse some
  (objects: processes, products, other experience models)
- for the purpose of
  why: characterisation, evaluation, prediction, motivation, improvement

Perspective:
- with respect to
  (focus: cost, correctness, defect removal, changes, reliability, user friendliness,...)
- from the point of view of
  who: user, customer, manager, developer, corporation,...

Environment:
- in the following context
  (problem factors, people factors, resource factors, process factors,...)

FIGURE 2. Goal Template.

1. Specify a set of goals based on the needs of the organisation and its projects. Determine what should be improved or learned. The process of goal definition is supported by templates like the ones shown in figure 2 [1]. By using these templates it is possible to define the goals in terms of purpose, perspective, and environment. The identification of subgoals, entities, and attributes related to the subgoals is made in this step.

2. Generate a set of quantifiable questions. Business goals are translated into operational statements with a measurement focus. Basili and Rombach [1] provide different sets of guidelines to classify questions as product-related or process-related. The same questions can be defined to support data interpretation of multiple goals.

3. Define a set of metrics that provides the quantitative information needed to answer the quantifiable questions. In this step, the metrics suitable to provide information to answer
the questions are identified and related to each question. Generally, each metric can supply information to answer several questions and sometimes a combination of metrics is needed to make up the answer of a question.

Once these steps are identified, data are collected and interpreted to produce an answer to the quantifiable questions defined to fulfil the goals of the organisation [1], [17].

1.4 Application of the Goal/Question/Metrics to the CMM

The CMM and the GQM can very easily be intertwined. The CMM defines one or more goals for each KPA as shown in figure 3. These goals can be used for the first step of the GQM. The CMM defines two distinct goals for the Requirements Management Key Process Area (KPA). The first one states the following:

"System requirements allocated to software are controlled to establish a baseline for software engineering and management use" [13].

It focuses on the control of requirements to set up a baseline, that is a kind of standard by which things are measured or compared. If the requirements are not controlled, there will be no clear picture of the final product, because the final product is based on the requirements. The second goal of the Requirements Management KPA states the following:

"Software plans, products and activities are kept consistent with the system requirements allocated to software" [13].

The main focus of this goal is the consistency between the requirements and any software product created from those requirements. This consistency would result in the design of the product required by the customer. The goals presented can be redefined by applying the goal template in figure 2 as follows. To Analyse the system requirements allocated to software for the purpose of establishing a baseline with respect to the control of the requirements from the point of view of academy and the software manager, in the context of the company where the Requirements Management is implemented. The second goal can be redefined as follows: to Analyse software plans, work products and activities for the purpose of consistency with the system requirements allocated to software from the point of view of academy and the software manager, in the context of the company where the Requirements Management is implemented.
The second step in the GQM paradigm is to generate a set of quantifiable questions. The questions have been produced by applying the guidelines for process related questions [1], analysing the goals of the Requirements Management KPA and its Key Practices [13] word by word, reading papers [8], [3], discussing with colleagues, and browsing web pages. For some questions a rationale will be given to better understand the meaning and/or the utility of a certain question.

**Questions for the First Goal of the Requirements Management KPA**

By analysing the first goal, a question arisen is: how can the requirements be controlled? And why should we control them? We know that it is not possible to specify in the beginning exactly what the customer wants. Neither it is possible to dictate the frequency or desirability of changes. The changes can come at the worst moment and impede our ability to finish a project with the available resources. The only possibility is to control the continuing definition of requirements as they change throughout the life cycle [16].

**TABLE 1: Questions and measures for the first goal of the Requirements Management KPA**

<table>
<thead>
<tr>
<th>Questions</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is the current status of each requirement?</td>
<td>Status of each requirement</td>
</tr>
</tbody>
</table>
| What is the level of the stability of the requirements? | # initial requirements
|                                               | # final requirements
<p>|                                               | # changes per requirement                   |</p>
<table>
<thead>
<tr>
<th>Questions</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why are the requirements changed?</td>
<td># initial requirements&lt;br&gt;# final requirements&lt;br&gt;# changes per requirement&lt;br&gt;# test cases per requirement&lt;br&gt;Type of change to requirements&lt;br&gt;Reason of change to requirements&lt;br&gt;Major source of request for a change to requirements&lt;br&gt;Phase where change was requested</td>
</tr>
<tr>
<td>What is the cost of changing the requirements?</td>
<td>Cost of change to requirements&lt;br&gt;Size of a change to requirements</td>
</tr>
<tr>
<td>Is the number of changes to requirements manageable?</td>
<td>Total # Requirements&lt;br&gt;# changes to requirements proposed&lt;br&gt;# changes to requirements open&lt;br&gt;# changes to requirements approved&lt;br&gt;# changes to requirements incorporated into base line&lt;br&gt;# changes to requirements rejected&lt;br&gt;The computer software configuration item(s) (CSCI) affected by a change to requirements&lt;br&gt;Major source of request for a change to requirements&lt;br&gt;Requirement type for each change to requirements&lt;br&gt;# requirements affected by a change</td>
</tr>
<tr>
<td>Does the number of changes to requirements decrease with time?</td>
<td># changes to requirements per unit of time</td>
</tr>
<tr>
<td>How are affected groups and individuals informed about the changes?</td>
<td>Notification of Changes (NOC) shall be documented and distributed as a key communication document&lt;br&gt;# affected groups and individuals informed about NOC</td>
</tr>
<tr>
<td>How many other requirements are affected by a requirement change?</td>
<td># requirements affected by a change</td>
</tr>
<tr>
<td>In what way are the other requirements affected by a requirement change?</td>
<td>Type of change to requirements&lt;br&gt;Reason of change to requirements&lt;br&gt;Phase where change was requested</td>
</tr>
<tr>
<td>Is the size of the requirements manageable?</td>
<td>Size of requirements</td>
</tr>
<tr>
<td>How many incomplete, inconsistent and missing allocated requirements are identified?</td>
<td># incomplete requirements&lt;br&gt;# inconsistent requirements&lt;br&gt;# missing requirements</td>
</tr>
<tr>
<td>Does the number of &quot;To Be Done&quot; (TBD) decrease with time?</td>
<td># TBDs in requirements specifications&lt;br&gt;# TBDs per unit of time</td>
</tr>
<tr>
<td>How are the requirements defined and documented?</td>
<td>Kind of documentation</td>
</tr>
<tr>
<td>Are the requirements scheduled for implementation into a particular release actually addressed as planned?</td>
<td># requirements scheduled for each software build or release</td>
</tr>
<tr>
<td>How many requirements are included in the baseline?</td>
<td># baselined requirements&lt;br&gt;phase when requirements are baselined</td>
</tr>
</tbody>
</table>

TABLE 1: Questions and measures for the first goal of the Requirements Management KPA
Any information on requirements can help to establish control. Especially important is to know the starting set and final set of requirements. To increase the control of the requirements, their status as well as their stability could be investigated (see questions 1 and 2 in table 1). The possible status of a requirement could be: new, analysed, approved, documented, rejected, incorporated into the baseline, designed, implemented, tested, etc. Requirements stability is concerned with the changes made in requirements, therefore a set of questions (see questions 3-9 in table 1) about requirements changes can be defined to refine the question 2. The level of requirements stability can be measured also by having information about the size of the requirements and by identifying problematic requirements (see questions 10-12 in table 1).

Once there is control over the requirements, a baseline must be established. Therefore some questions about how the requirements are documented, and how many of them are included in the baseline, are defined (see questions 13-15 in table 1).

Table 1 shows all the questions defined and the measures proposed to give information to answer questions. Please observe that some of these questions are also used for the second goal of the Requirements Management KPA, for instance, questions 13 and 14.

**Questions for the Second Goal of the Requirements Management KPA**

The purpose of the second goal is mainly to keep consistency between the requirements and the software project, therefore it is suggested to keep traceability among the software documents. Traceability between requirements and the software project facilitates the analysis of the effects of a software change and reduces the effort to locate the causes of a product failure. Tracking the requirements and changes made to requirements can help to maintain traceability (questions 1-7) among the requirement documents.

**TABLE 2**: Questions and measures for the second goal of the Requirements Management KPA.

<table>
<thead>
<tr>
<th>Questions</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does the software product satisfy the requirements?</td>
<td># initial requirements</td>
</tr>
<tr>
<td></td>
<td># final requirements</td>
</tr>
<tr>
<td></td>
<td># test cases per requirement</td>
</tr>
<tr>
<td></td>
<td>Type of change to requirements</td>
</tr>
<tr>
<td>What is the impact of the changes to requirements on the software project?</td>
<td>Effort expended on Requirements Management activity</td>
</tr>
<tr>
<td></td>
<td>Time spent in upgrading</td>
</tr>
<tr>
<td></td>
<td># documents affected by a change</td>
</tr>
<tr>
<td>What is the status of the changes to software plans, work products, and activities?</td>
<td>Status of software plans, work products, and activities</td>
</tr>
<tr>
<td>Are the requirements scheduled for implementation into a particular release actually addressed as planned?</td>
<td># requirements scheduled for each software build or release</td>
</tr>
<tr>
<td>How are the requirements defined and documented?</td>
<td>Kind of documentation</td>
</tr>
</tbody>
</table>
When the changes to requirements have been implemented, other documents are most probably affected by this change. Therefore it is suggested to check the status of software plans, work products, and activities, which could be "identified, evaluated, assessed, documented, planned, communicated to affected groups and individual, and tracked to completion".

### 1.5 Measures for the goals of the requirements management KPA

The third step of the GQM is to define sets of metrics that provide the quantitative information necessary to answer the questions. In this paper, only a set of measures is presented, which is shown in tables 1 and 2. There are overlaps among the questions for the two goals and among the measures. The same measure can be used to give information to answer different questions. Some of the measures are suggested by the SW-CMM (status of allocated requirements, change activity, and cumulative number of changes to allocated requirements). Other possible measures are the number of test cases assigned to each requirement, by which it is possible to check how many requirements are verifiable; the size and the cost of a change request which could make it possible to predict the project cost and the schedule. Other measures on requirements are the ones recommended by [9] who measure attributes of requirements like ambiguity, modifiability, priority, etc. However, these are measures on the requirements and not on the requirements management process therefore these are out of the scope of this paper.

Once the three steps of the Goal Question Metric paradigm are defined, an organisation has to determine ranges for "good data" (for instance a company could accept no more than three change requests per week). Data must be collected and compared to the "good data", and eventually improvement actions are taken. A comparison of the data for the actual project with the once collected for previous projects will provide a baseline for the requirements and give meaning to the measures.

### 1.6 Testing the measures in a company

At the time of writing, the measures are being tested at Ericsson Erisoft AB in Umeå, Sweden, which is a medium-size company. People at the company used SW-CMM and improved their software process to level 2 but caused by a re-organisation they decreased their CMM matu-
A previous study made in this company [4] shows that people at the company apply some basic measurements but they do not perform any significant evaluation of the data collected.

The test of the measures is conducted following the guidelines in [18] about how to conduct an experiment; the first step of the experiment process (definition of the experiment) has been done.

**Object:** Requirements Management activity at Ericsson Erisoft AB in Umeå, Sweden

**Purpose:** Evaluate the impact of the measures to improve the Requirements Management activity

**Quality focus:** Control of the requirements

**Perspective:** Academy

**Context:** Medium-size company

While planning the study (which is the second step of the experiment process suggested in [18]), the author is learning the company’s organisational structure, and the software processes used in the company. The software processes used are PROPS as management process and RUP (Rational Unified Process) [10] as software development process. The author is also going to map the terminology used by people in the company to the CMM terminology. Some data from studying one particular increment have been collected (a project can have many increments and each increment is a refinement of the software product). A preliminary conclusion of this study is that the domain analysis is not done very deeply and the requirements are added and changed during the software life cycle.

### 1.7 Concluding remarks and future directions

An application of the GQM to the Requirements Management KPA has been reported. The result of the application of the GQM to the CMM, is a list of measures for each KPA.

The set of questions and measures presented should be tailored to the particular organisation. All level 1 companies interested in improving the Requirements Management activity are suggested to select a subset of these measures especially useful to start with. For instance, a level 1 organisation has most probably poorly defined requirements. The visibility of the process is very low at this level, and it is difficult to measure the process. Therefore, it is suggested that they count the number of requirements and changes to those requirements to establish a baseline. Other level 1 companies (like the ones referred in [5]) need to document the requirements before starting to measure. A company like Ericsson-Erisoft in Umeå (who has improved the process through the CMM in the past), can collect all the data regarding the change activity. This is possible because information about change requests is stored in reports and database.

The measures produced in this paper provide the organisation with improved visibility and better insight into the Requirements Management activity, improving the software process a small step towards the goal of having a repeatable process. The measures can be used
in quantifying the amount of changes to requirements and to predict the cost for a change, helping to control requirements and changes to requirements. If the process is repeatable, more information on requirements can and should be collected, such as type of each requirement (database requirement, interface requirement, performance requirement, etc.) and change requests to each type. In general, the metrics collection will vary with the maturity of the process.

This work has been done with the aim of joining the assessment and measurement based methodologies mentioned above. The aim has been addressed by analysing the Requirements Management KPA of the SW-CMM and its key practices [13], and applying the GQM paradigm to it. The underlying assumption in the paper is that it is easier to focus the measurement and improvement activities on a specific process area, rather than trying to measure and improve all the process areas at once. The results are relevant to areas such as Requirements Management, Software Measurement, Software Process Improvement, Software Quality, etc.

The measures obtained, will be used for the elicitation of requirements information in the before-mentioned company. For each measure proposed, improvement actions will be suggested to help decision makers in their job. After these actions are taken, an enterprise can follow the approach described in "Application of Metrics in Industry" [14], which suggests to cycle four steps (analyse, act, metricate, and improve) to continue with the improvement of the KPA. Based on the data collected, suggestions for improvement of the Requirements Management activity will be given. The measurements will be automated as far as possible because personnel treat measurement as boring. Finally, application of the GQM to all the KPAs of the CMM levels will be considered.

1.8 Acknowledgements
Special thanks to my supervisor Jürgen Börstler for his suggestions and contribution to this paper and to all the people who have supported and contributed to the writing of this work.

1.9 References


Abstract
The goal of this position paper is to perform a non-empirical validation of the "number of requirements" measure, which can help to control the Requirements Management process. The reason to validate this measure is to contribute to the lack of rigorous measures validation in the literature. The aim has been addressed by applying two definitions of software measure validation to the measure. However the measure is still not validated because of the impossibility to prove that the representation condition is satisfied both theoretically and empirically. Several questions are arisen and discussed and they confirm the immaturity of the measures validation field.

General Terms
Management, Measurement.

Keywords
Measures validation, theoretical validation, Requirement Management.
2.1 Introduction

Managing requirements is an important activity of quality improvement. It is a way to control the continuous definition of requirements as they change through the software lifecycle. The control over the requirements helps in anticipating and responding to requests of change. Software measurement can help us in controlling the requirements process, in quantifying the amount of change to requirements and in predicting the cost of a change. In general software measurement provides guidance to the management activities. Numerous software measures for the Requirements Management activities are found in the literature. Unfortunately many of these are not validated rigorously, thus there is still the need to validate software measures. Two kinds of validations are necessary for this purpose: theoretical validation by which we demonstrate that a measure is properly representing the attribute it claims to measure. This is a basic form of validation and a prerequisite to the second kind of validation, the empirical one. Empirical validation proves that the measure is useful i.e. the measure is connected to other variables in expected ways. We choose to discuss about theoretical validation because it is a prerequisite to empirical validation and because too little effort has been done in this field. Several people have tried to define rules for validation but these rules have been criticised and there are only few measures validated in a theoretical way and documented. Therefore the goal of this position paper is to discuss validation techniques and to validate the measure "number of requirements" and similar measures (from now on we will use the symbol \# to denote "number of"). Several questions arise after the application of the validation definition, for instance: is it important to show that the representation condition is satisfied? Is there a better way to perform a non-empirical validation?

This paper is organised as follows: section 2 contains the motivation of supporting measurement for managing requirements. Section 3 presents the definition of measure validation with different perspectives, and the application of a validation process. Finally conclusions and future works are presented.

2.2 Measures to improve the requirements management process

A set of measures for the Requirements Management Key Process Area (KPA) of the Capability Maturity Model for software (SW-CMM) [7] has been proposed in [5]. This set is the result of an analysis of the Requirements Management KPA of the SW-CMM and its key practices and the application of the Goal Question Metrics (GQM) paradigm [1] to it. We have defined a total of 38 measures, 33 for the first goal of the Requirements Management KPA and 12 for the second goal. A sample of these can be seen in table 1. Some of the measures are used to furnish the quantitative information necessary to answer different questions. These measures provide an organisation with increased visibility and better insight into the Requirements Management activity, improving the software process a small step towards the
goal of having a repeatable process. The measures can be used in quantifying the amount of changes to requirements and to predict the cost of a change, helping to control requirements and changes to requirements.

2.3 Measure validation

Our set of 38 measures has not yet been validated either theoretically or empirically. Before applying theoretical validation to some of these measures, we would like to discuss some definitions.

**Internal-external validation**

Fenton and Pfleeger [3] define measure validation as follows: "validating a software measure is the process of ensuring that the measure is a proper numerical characterisation of the claimed attribute by showing that the representation condition is satisfied". This definition is also known as internal validation or validation in the narrow sense [3]. Internal validation
of software measures is based on a homomorphism between the empirical world and the nu-
merical world. We can also say that internal validation of software measures is done without
predicting an external attribute. External validation is done by showing that an external at-
ttribute is a function of an internal one [9]. We prove that an external attribute X verifies the
equation X=f(Y) where Y is an internal attribute. In general the connection between external
and internal attributes is built through a prediction model [3]. Predictive validity can be de-
termined only by performing an empirical validation. When performing empirical validation
however, the connection between internal attribute values and the resulting external attribute
values are seldom sufficiently proven. There are two reasons for this [3], 1) it is difficult to
perform controlled experiments and confirm relationships between attributes and 2) there is
still little understanding of measurement validation and the proper ways to demonstrate these
relationships.

**Theoretical validation**
Theoretical validation allows us to say if a measure is valid respect to some defined criteria.
Several attempts to define properties of a sound measure are present in the literature, (e.g. [2],
[4], [6], [10]), but they have all been criticized. [4] (which summarise and extend [6] and [10])
and [2] both make a distinction between theoretical and empirical validation. Since we are in-
terested in theoretical validation, we note that in [4], the definition of theoretical validation
identifies four criteria that need to be satisfied by a valid measure. While in [4] the represen-
tation condition is one of these criteria, in [2] it is the main focus of their validation definition.
But as stated in [8] and [9] the proof that the representation condition is satisfied can only be
empirical by its nature. Therefore we find it somewhat confusing to denote the definitions of
theoretical validations find in [2] and [4] theoretical, because, to our knowledge, they use a
property that can only be proven empirically.

**Validating requirements management measures**
The process of defining software measures starts by observing real world entities and identi-
fying properties of these entities we would like to investigate. The entities we are observing
are software activities related to the management of requirements. These entities can be or-
ganised hierarchically as shown in figure 1. On the top of this hierarchy we find the Require-
ments Management process. The structure shown in figure 1 is not a complete tree, more
leaves can be defined such as baselined requirements, open change requests, approved change
requests, etc. The entities shown in figure 1 are the ones we regard as being the most inter-
esting for our discussion. They can have several attributes associated to them. The ones nec-
essary in order to reach the two predefined goals of the Requirements Management KPA are
stability, traceability, consistency etc. These are external attributes and the measures defined
for these attributes can only be validated empirically, by doing experiments (see paragraph 3.1
for an explanation of the relationship between internal and external attributes).
The association of the entities and attributes to the measures for the Requirements Management process is shown in table 1. The relationship between entity-attribute and attribute-measure is many to many values.

For reasons of space we cannot validate here all the 38 measures proposed in [5]. As an example we will theoretically validate the "# requirements". We believe that the results of the validation are also applicable for measures like # initial requirements, # final requirements, # changes to requirements etc. because of similarity among these measures. The "# requirements" is a simple count of the requirements. This includes all technical and non-technical requirements as originally provided by the customer. This measure will be used in conjunction with the # initial and final requirements as well as the # changes per requirement to assess the level of requirements volatility. This measure can also help to provide insight into why requirements are changed and if the software product satisfies the requirements. In order to produce a valid software measure we choose the "key stages of formal measurement" that is sketched out in [3].

1. Identify attributes for some real world entities: the entity associated to our chosen measure "# requirements" would be requirements specification. An internal attribute of the requirements specifications is its size. It would be more interesting to measure the stability of the requirements but unfortunately it is difficult to determine empirical relations for the attribute stability. The validity of the measures connected to this external attribute can only be established by performing an external validation. The reason of choosing the size of requirements specification is because we think that there is a cause effect relationship between the attributes size and stability: if the size of a requirements specification decreases or increases a lot in time, then the requirements are unstable.
2. Identify empirical relations for the attribute: several binary relations can be established for the attribute "size", e.g. "bigger", "smaller", "equal to", etc.

3. Identify numerical relations corresponding to each empirical relations: the corresponding numerical binary relations to the empirical ones can be $>$, $<$, $=\ldots$, etc.

4. Define mapping from real world entities to numbers: For the mapping we have several options, we can choose non-mathematical symbols, natural numbers, integers, and real numbers. The domain of the mapping (real world) is made by all possible requirements specifications and it is a finite domain. The range of the mapping (mathematical world) is the set of natural numbers $n$ such that: $0 < n < \ldots$. The range is a finite set. One mapping rule could be for instance to count only atomic, non-empty requirements.

5. Check that numerical relations preserve and are preserved by empirical relations: this is the most important step of the validation process since here we have to show if the representation condition is satisfied. This is hard to prove, we are not able to prove that the size of a requirements specification $A$ is smaller than the size of a requirements specification $B$ if the number of requirements of $A$ is less than the number of requirements of $B$. In short, we cannot prove the following: $\text{size}(A) < \text{size}(B) \Rightarrow \#\text{requirement}(A) < \#\text{requirement}(B)$. If we empirically compare the size of the requirements specification $A$ and $B$, it might happen that in some particular context the representation condition is satisfied. This leaves us with the open question: how many requirements specifications should we compare to be able to say that the representation condition is satisfied?

By following the "key stages of formal measurement" we cannot say that the measure "$\#\text{requirements}$" is a valid measure of the size of requirements specification, because we cannot prove that numerical relations are preserved by empirical relations. Even empirically we might not be able to show that the representation condition is satisfied because the size of two atomic requirements might be different from each other. In general, it is not sure that the "$\#\text{requirements}$" is a good measure of the attribute size, because in requirements specifications there can be many small atomic requirements or few big atomic requirements. These atomic requirements could be of different types, non comparable, e.g. they can be written by using different notations. Therefore we might need to further decompose the $\#\text{requirements}$ in measures such as $\#$ use cases, $\#$ bubbles in a data flow diagram etc. Using $\#\text{requirements}$ as a measure of the size of requirements specifications might be misleading. In fact there is still little consensus on how to measure the size of requirements specifications.

We will now apply the method of theoretical validation defined in [4] to validate the measure "$\#\text{requirements}$". 

1. For an attribute to be measurable, it must allow different entities to be distinguished from one another. In our case, there must exist two requirements specifications, for which a counting of requirements would result in different values. This property is (intuitively) satisfied.

2. A valid measure must obey the representation condition. This criterion is similar to the fifth property of the "key stages of formal measurement". As stated earlier, we cannot prove the validity of this criterion.
3. Each unit of an attribute contributing to a valid measure is equivalent. The attribute “size” does not imply directly a measurement scale. But if we identify empirical relations (that are needed in order to verify the representation condition) for this attribute, we come up with binary relations like “bigger”, “smaller”, and “equal to”. These relations imply an ordering of the entities being measured. We could identify empirical relations such as “equal to” or “different” which do not imply an ordering of the entities, but we would lose some properties of the attribute size.

4. Different entities can have the same attribute value. This means that the “# requirements” must allow different requirements specifications to have the same # requirements value. This property is (intuitively) satisfied.

By applying the theoretical validation proposed in [4] we again run into the problem of demonstrating that the representation condition is satisfied and we are therefore not able to validate the measures.

2.4 Conclusions and future work

In this paper we have applied two non-empirical validation definitions to the “# requirements” measure. However, none of the two methods applied has given good results, leaving the “# requirements” still non-validated. This is due to the impossibility of verifying the concept of representation condition. This concept is the most difficult property to verify among the ones shown earlier. But we doubt the importance of verifying it. In the case of size of requirements specification and in the context of measuring the process, we are not interested in comparing the sizes of two different requirements specifications rather to see the variation of the size of one requirements specification over time.

The concept of representation condition is part of the definitions of theoretical validation in [2] and [4]. Both make a distinction between theoretical and empirical validation. As stated in [8] and [9] the proof that the representation condition is satisfied can only be empirical by its nature. Therefore we find it somewhat confusing to denote the definitions of theoretical validations find in [2] and [4] theoretical, because, to our knowledge, they use a property that can only be proven empirically.

In the case of validating the # requirements as a measure of the size of requirements specifications, we also doubt the possibility of proving that the representation condition is satisfied empirically. How many requirements specifications should we compare to be able to say that the representation condition is satisfied? We think that an empirical validation would require a large amount of data and a manifold of different data and therefore can rarely be conducted in the proper way. Other questions are arisen here: among the methods present in the literature to theoretical validate a software measure what is best in order to produce sound software measures? Is there a better way to perform a non-empirical validation? What can be done when it is not possible to conduct an empirical validation of the measures?

Sadly, few measures have been validated and documented. One reason is that there still is a non-widely accepted way of validating a measure as stated in [8] and [9]. As future work we intend further investigate theoretical validation techniques and to validate empirically the 38
measures proposed in [5]. Furthermore, we intend to construct prediction models, which include our measures, and to perform experiments for the purpose of testing the prediction models created.

2.5 References


Abstract

Requirements management measures can help us to control changing software requirements and estimate the costs of changing requirements. The goal of this paper is twofold. First we describe a set of requirements measures and validate these measures applying two definitions from the literature. After that we describe a case study using these measures to estimate the cost of changes to requirements. Although we were not able to collect sufficient data to draw statistically significant conclusions, we present some lessons learned.

Keywords:
Requirements Management, Measure Validation, Theoretical, Empirical, Internal, External Validation, Case Study.
3.1 Introduction

Carefully developed software requirements are a key issue for project success [29]. The cost of correcting an error after system delivery is orders of magnitude higher than the cost of correcting a similar error during the requirements analysis phase [21].

Since requirements change frequently, even during the development, it is important to control those changes to be able anticipate and respond to change requests [24]. Requirements development is a learning process rather than a gathering process. Therefore, it is naïve to believe that we can specify exactly what a customer wants at the beginning of a project. The best we can do is to carefully monitor and control all requirements throughout the software life cycle.

Requirements management is an activity performed in parallel with requirements elicitation, analysis, documentation, and validation (see Figure 1). It is the process of monitoring and documenting changes to requirements, tracing these changing to related work products, and communicating this information across the project team. Requirements management practices ensure that unanticipated changes can be controlled throughout the project life cycle. Without these practices high quality software is difficult to achieve.

![FIGURE 4. Requirements Engineering Process (adapted from [13]).](image)

The management of the requirements is connected to other areas of software engineering like software maintenance, configuration management, and change management. Software maintenance is triggered by change requests from customers or marketing requirements. A change request can be a change to requirements and can be filed before product delivery, i.e. during the software development. New versions of software systems are created as they change, and this makes the systems evolve. Configuration management involves the develop-
ment and application of procedures and standards to manage an evolving software system. Configuration management aims to control the costs and effort involved in making changes to a system. Requirements management and configuration management are two important key process areas of the Capability Maturity Model for Software (SW-CMM) and they may be seen as part of a more general quality management process [20].

**TABLE 1:** The 38 Measures Defined in [14]

<table>
<thead>
<tr>
<th>Requirements Management Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. # affected groups and individuals informed about NOC</td>
</tr>
<tr>
<td>2. # baselined requirements</td>
</tr>
<tr>
<td>3. # changes per requirement</td>
</tr>
<tr>
<td>4. # changes to requirements approved</td>
</tr>
<tr>
<td>5. # changes to requirements incorporated into base line</td>
</tr>
<tr>
<td>6. # changes to requirements open</td>
</tr>
<tr>
<td>7. # changes to requirements per unit of time</td>
</tr>
<tr>
<td>8. # changes to requirements proposed</td>
</tr>
<tr>
<td>9. # changes to requirements rejected</td>
</tr>
<tr>
<td>10. # documents affected by a change</td>
</tr>
<tr>
<td>11. # final requirements</td>
</tr>
<tr>
<td>12. # incomplete requirements</td>
</tr>
<tr>
<td>13. # inconsistencies</td>
</tr>
<tr>
<td>14. # inconsistent requirements</td>
</tr>
<tr>
<td>15. # initial requirements</td>
</tr>
<tr>
<td>16. # missing requirements</td>
</tr>
<tr>
<td>17. # requirements affected by a change</td>
</tr>
<tr>
<td>18. # requirements scheduled for each software build or release</td>
</tr>
<tr>
<td>19. # TBDs in requirements specifications</td>
</tr>
<tr>
<td>20. # TBDs per unit of time</td>
</tr>
<tr>
<td>21. # test cases per requirement</td>
</tr>
<tr>
<td>22. Cost of change to requirements</td>
</tr>
<tr>
<td>23. Effort expended on Requirements Management activity</td>
</tr>
<tr>
<td>24. Kind of documentation</td>
</tr>
<tr>
<td>25. Major source of request for a change to requirements</td>
</tr>
<tr>
<td>26. Notification of Changes (NOC) shall be documented and distributed as a key communication document</td>
</tr>
<tr>
<td>27. phase when requirements are baselined</td>
</tr>
<tr>
<td>28. Phase where change was requested</td>
</tr>
<tr>
<td>29. Reason of change to requirements</td>
</tr>
<tr>
<td>30. Requirement type for each change to requirements</td>
</tr>
<tr>
<td>31. Size of a change to requirements</td>
</tr>
<tr>
<td>32. Size of requirements</td>
</tr>
<tr>
<td>33. Status of each requirement</td>
</tr>
<tr>
<td>34. Status of software plans, work products, and activities</td>
</tr>
<tr>
<td>35. The computer software configuration item(s) (CSCI) affected by a change to requirements</td>
</tr>
<tr>
<td>36. Time spent in upgrading</td>
</tr>
<tr>
<td>37. Total # Requirements</td>
</tr>
<tr>
<td>38. Type of change to requirements</td>
</tr>
</tbody>
</table>
In a survey of 4000 European companies [28] it was found that the management of customer requirements was one of the main problem areas in software development. Software measurement can help us in providing guidance to the requirements management activities by quantifying changes to requirements and in predicting the costs related to changes. Numerous software measures for the requirements management activities have been proposed in the literature (see [4] and [23]). However, few of these measures have been validated. Furthermore, there are only few empirical studies in this area. In our previous work [14] we analysed the key practices defined in the Requirements Management Key Process Area (KPA) of the SW-CMM [20]. By means of the Goal Question Metrics (GQM) paradigm [1] we defined a total of 38 measures, shown in table 1. Organisations are supposed to select suitable measures from this list, depending on their practices, processes and maturity. In general it is suggested to keep the set of data collected small [2]. Immature organisaiton have poor visibility of the requirements management process or/and poorly defined requirements. Most of the measures in [14] are therefore not applicable. Immature organisations should document their requirements and after that they can start counting the total # requirements and changes to those requirements to establish a baseline. Organisations at higher maturity levels will likely have established processes to handle change requests. They can (and should) therefore collect further data, such as type of requirement (database, interface, performance requirement, etc.). In general the metrics collection will vary with the maturity of the process.

In this paper we will validate a subset of our set of 38 measures (shown in Table 2), which will later be used in a case study performed in a class project. Unfortunately, we were not able to collect enough data to draw statistical conclusions. However, some lessons learned are presented.

The reminder of this paper is organised as follows. Section two introduces basic software measurement theory. Section three summarises previous work on measures validation. In the fourth section we perform a theoretical validation of the measures. In section five we describe a case study with the purpose of showing that effort estimations using historical data are better than intuitive estimations.

### 3.2 Software measurement

In software measurement theory, an entity is an object, an attribute is a property of an entity, a measure is the number or symbol assigned to an entity in order to characterise the attribute. Software measures can be direct or indirect. Direct measurement of an attribute of an entity involves no other attribute or entity. Indirect measurement of an attribute of an entity involves other attributes or entities. Attributes can be internal or external. Internal attributes are those that can be measured by examining the entity on its own, separately from its behaviour. External attributes are measured by means of other attributes. Software measurement leads to the following advantages:

- increased understanding and control of software development process,
- increased capacity to improve the software development process,
- more accurate estimates of software projects cost and schedule,
more objective evaluation of changes in techniques, tools and methods,
more accurate estimates of the effects of changes on project cost and schedule,
decreased development costs due to increased productivity and efficiency,
improved customer satisfaction and confidence due to higher productivity quality [30].
low cost of software measurement activities. The amount of time spent by a software development team on a measurement program is about 2% of their time reduced to 1% when the team gets experienced.

Although there are many advantages, measurement can also lead to problems.

Honesty of measurement: human behaviour will be changed. When measuring the effort to evaluate personnel productivity, the developers tend to be vague to apply metrics and may report inaccurate data to avoid a reprimand.

Fear of measurement, resistance to measurement, and ethical use of measures. These are social issue and software engineers are poorly equipped to deal with them [5].

Emphasis will be layed on what is being measured, what is not being measured will gradually be ignored.

Besides the disadvantages of software measurement, other factors should be kept in mind if we want to perform a successful measurement program: feedback sessions with people collecting data and cost benefit analysis [30].

3.3 Measures validation

Several definitions of measures validation are present in the literature. The probably most recognised is the internal-external validation. Fenton and Pfleeger [10] define measure validation as follows: “validating a software measure is the process of ensuring that the measure is a proper numerical characterisation of the claimed attribute by showing that the representation condition is satisfied”. This definition is also known as internal validation or validation in the narrow sense. Internal validation of software measures is based on a homomorphism between the empirical world and the numerical world.

Through the representation condition we demonstrate that the relationship between the measures and the internal attributes is valid in the empirical world. Suppose we want to measure the temperature of two glasses of water A and B. In the real or empirical world we feel the temperature and decide that for example A is warmer than B. In the mathematical world we choose a numerical relation associated to the empirical relation “is warmer”. The numerical relation would in this case be “>”. Then we choose a temperature system (Celsius, Fahrenheit etc.) and finally we associate numbers to the temperature of each glass of water. Suppose that in the numerical world TC(A) = 25 and TC(B) = 37, where TC(X) is the temperature in Celsius degree of X. The representation condition would not be satisfied because it does not correspond to what happen in the real world.

An example for requirements is the following: size(S) < size(T) if and only if #requirement(S) < #requirement(T) where S and T are two Software Requirements Specification (SRS) documents; size is the attribute of the entity requirement (maybe it is the entity requ
As explained in [15], the relationship above can be proven only empirically. One way to internally validate a measure, is to follow the “key stages of formal measurement” [10], shown in Figure 1.

External validation is done by showing that a measure is 1) internally valid and 2) a component of a prediction system [10]. The definition given by [32] of external validation states that we perform an external validation if we prove that an external attribute (or external variable) is a function of an internal one. We prove that an external attribute $X$ verifies the equation $X = f(Y)$ where $Y$ is an internal attribute. For instance cost = $f$(LOC) where “cost” is an external attribute of the entity “program” and LOC (Lines Of Code) is an internal attribute of the same entity. In general the connection between external and internal attributes is built through a prediction model. However the distinction between internal, external attributes, direct and indirect measures is sometimes not clear. According to [34], external attributes are mostly indirect measures and internal attributes are mostly direct measures.

Predictive validity can be determined only by performing an empirical validation. When performing empirical validation however, the connection between internal attribute values and the resulting external attribute values are seldom sufficiently proven. As stated in [10], there are two reasons for this; 1) it is difficult to perform controlled experiments and confirm relationships between attributes and 2) there is still little understanding of measurement validation and the proper ways to demonstrate these relationships.

The same concepts of internal and external validation are used in the wider definitions of theoretical and empirical validation [12]. Theoretical validation allows us to say whether a measure is valid with respect to some defined properties (the ones listed in Figure 1). The terms “theoretical validation” are used in [12] in a broader sense, they include the validation of the measurement instruments and of the data collection procedures. In the next section we will apply this theoretical validation but we will focus only on the properties of the entities, attributes, and measures.

When we perform an empirical validation we verify that measured values of attributes are consistent with values predicted by models involving the attribute [12]. For example measured values of cost are consistent with the values obtained by a function $f$(LOC). As stated in [15], it is not possible to theoretically validate a measure without performing an empirical study, because the representation condition can be only proven empirically [10].

Briand et al. ([5], [6]) support theoretical and empirical validation. Theoretical validation is based on the construction of an empirical relational system and a formal relational system. The properties of the empirical system should be preserved by the formal system if the measure is valid. Their definition of theoretical validation is taken from the classical definition in measurement theory. Empirical validation is based on the proof that internal attributes are connected to external attributes. In other words with empirical validation we prove that a measure is useful, i.e. that it is connected to a goal.
We can also validate empirically a direct measure by asking the participants of an empirical study whether a mapping to a value captures their understanding of the attribute. This kind of validation is based on interviews with the people supplying the data. It is an interactive data validation process [2].

Shneidewind [27] defines a measures validation process that integrates quality factors, metrics, and quality functions. The validation process includes six validity properties that support quality aspects like assessment, control, and prediction. Another approach to validate software product measures is the GQM/MEDEA (MEtric DEfinition Approach) [8]. It combines the GQM, the approach in [6] and [7], and some guidelines to perform experiments in [34].

Several attempts to define properties for sound measures are present in the literature (e.g. [6], [12], [18], [33]). Weyukers [33] identified nine properties which are useful to evaluate syntactic software complexity measures. Her work has raised an intense discussion about validation ([12], [32], [7], [19]). We can conclude that there is not yet a general accepted way of validating software measures. As stated in [8], the field of measures validation is still in a state where terminology and definitions have to become more consolidated.

3.4 Theoretical validation of requirements management measures

In this section we describe the measures we will use in the case study (see Table 2). We apply the “key stages of formal measurement” [10] and the theoretical validation [12] shown in Figure 1.

By following the first five properties we construct an empirical system and a mathematical system. After that we define a mapping from these two systems. A mapping is a function from the empirical world to the mathematical world, therefore it has a domain and a range. The real world is the domain of the mapping and depends on the entity we are measuring. It is usually made by the set of different instances of the entities. The mathematical world is the range of the mapping. To determine the range of the mapping we have several options. We can choose non-mathematical symbols, natural numbers, integers, real numbers, etc. When defining a mapping we have to choose the rules of the mapping (the way we measure). We regard these rules to be context dependent, they depend on the environment where the data is collected. For each measure in Table 2 we will describe the mapping rules used in the case study.

By analysing the theoretical validation (shown in Figure 1), the attributes of an entity can have different measures associated to them and each measure can be connected to different units. Suppose we want to measure the attribute “size” of the entity “room”. The length or the area can be two measures of the size of the room. The length can be measured with the units centimetre or feet or the number of people that can fit in the room side by side. According to property 8, attributes are independent from the unit we choose to measure them, and any definition of an attribute that implies a particular measurement scale is invalid [12]. The ar-
tribute determines the scale type. In the traditional measurement theory only interval and ratio scale measures can have different units [12]. In order to choose the appropriate statistical method in the empirical study, we need to associate the scale type we will use to each measure. According to [6], knowing the scale type of a measure with absolute certainty is unrealistic in the majority of the cases.

Our criticism on the “key stages of formal measurement” is that in some cases it assumes that the values of the measures are numerical while our measures can assume non numerical values (like requirements status, type of requirements, etc.). In those cases, the mathematical world will be made by non-numerical symbols and we will be able to do only a restricted number of operations. The properties 1, 2, and 3 are shown in Table 2. These properties correspond to the definition of attributes, empirical, and numerical relations. We need also to determine the domain, range, scale of each measure to verify property 4. The properties 5 and 7 are equivalent (the representation condition) and will be applied only once when possible. The properties 6, and 9 are intuitively satisfied for our ten measures. Property 8 will be applied on few measures because the other measures are similar to those validated through property 8. The measures in Table 2 are direct measures.

By applying the first property of Figure 1 we identify real world entities and the properties of the entities we would like to investigate. Our main goal is to measure the requirements management. However, the management of requirements includes several activities and to define empirical and numerical relations we need to define more refined entities. These entities can be organised hierarchically as shown in Figure 4. On the top of this hierarchy we have the

---

<table>
<thead>
<tr>
<th>Key stages of formal measurement [10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify attributes for some real world entities.</td>
</tr>
<tr>
<td>2. Identify empirical relations for the attribute.</td>
</tr>
<tr>
<td>3. Identify numerical relations corresponding to each empirical relations.</td>
</tr>
<tr>
<td>4. Define mapping from real world entities to numbers.</td>
</tr>
<tr>
<td>5. Check that numerical relations preserve and are preserved by empirical relations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theoretical validation [12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. For an attribute to be measurable, it must allow different entities to be distinguished from one another.</td>
</tr>
<tr>
<td>7. A valid measure must obey the representation condition.</td>
</tr>
<tr>
<td>8. Each unit of an attribute contributing to a valid measure is equivalent.</td>
</tr>
<tr>
<td>9. Different entities can have the same attribute value.</td>
</tr>
</tbody>
</table>

**FIGURE 5. Measures Validation Properties.**
Requirements Management which is a process entity. As we go down in the hierarchy, we find more refined entities, which are the ones measured in practice. By applying the GQM and defining goals and subgoals we would construct a similar hierarchical structure.

The structure shown in Figure 5 is not a complete tree, more leaves can be defined such as RM phases, RM tools, etc. The entities shown in Figure 5 are the ones more closely connected to the goals of the Requirements Management KPA of the SW-CMM. Among them are “atomic requirement”, “requirements specifications” and changes to requirements.

**Where?**

There exist some definitions of atomic requirement (see for example [26]). Most references try to make atomic requirements context free i.e. stand alone. Different people may have different perceptions as to how much to split requirements to get an atomic indivisible requirement. We are aware of the risks of splitting a requirement, the context of the requirement could be lost, and the specifications could become unreadable. However, for our purposes, the “atomicistic” view of requirements eases our discussion. We assume that an atomic requirement is a (numbered) paragraph containing the word “shall” or “must”.

During the software development process, each requirement goes through different states as shown in Figure 5. The process shown in the Figure is similar to a change requests process. (shown in the user manual of the ClearDDTS tool). Some of the states in the Figure (subcontracted, reused, and cancelled) have been added after the execution of the case study because we understood the importance of these states during the execution of the study. Among these states there is a partial order. We can count the requirements in each of the phases to get a global view of the status of the SRS.
The entities we have mentioned above, can have several attributes associated to them. The ones we would like to measure are stability, traceability, consistency etc. These attributes are relevant to determine whether the general goals of the Requirements Management KPA are reached [20]. These are external attributes and the measures defined for these attributes can only be validated empirically.

**Size of software requirements specification**

One of the entities in Figure 4 is software requirements specification (SRS). A SRS can be a document or a database that contain requirements. There are different kinds of SRS documents, depending on the degree of formality. The informal SRSs are usually used to communicate with the customer, while the formal SRSs are used to communicate with the developers. As pointed out in [13], there is no standard name for requirements documents. For each requirement in the requirements document we suggest to include additional information like the state of the requirement, the rationale, the source (the person that has requested the requirement), the priority, etc. because this information can help in doing estimations of cost or stability of requirements.
By applying the first property of the “key stages of formal measurement” (see Figure 1) to the entity SRS we can identify internal attributes, for instance size. The reason for choosing this attribute is because we think that there is a cause-effect relationship between the attributes size and stability of the requirements: if the size of a requirements specification changes irregularly in time, then the requirements might be unstable. We intend to prove this hypothesis in future works through an empirical study. The validity of the measures connected to this external attribute can only be established by performing an external validation. The attribute size of SRS can be measured by counting the requirements. This count can be done at different moments of the software development process, i.e. initial, current, and final moment.

The scientific community is not yet in agreement on how to deal with units and scales of a software measure. The attribute size does not imply directly a measurement scale. However, if we identify empirical relations for this attribute, we find binary relations like bigger, smaller, equal to, etc. These relations imply an ordering of the entities being measured. We could identify empirical relation (such as equal to or different) which do not imply an ordering of the entities, but we would lose some properties of the attribute size. Property 8 in Figure 1 says that it is not important which unit we choose if the measure is valid i.e. if the representation condition is satisfied. According to [5, 6] instead interesting properties of the entity would be lost if we choose a measurement unit belonging to a lower scale. If we choose a unit belonging to the nominal scale (such as “empty requirements”, “non empty requirements”) we can lose some properties of the attribute size.

The scale we choose for the attribute size is a ratio scale because there are different ways to measure this attribute. In fact, the size of SRS can be measured by counting the # pages, # rows and similar measures.

**Total Number of Requirements**

The total # requirements is a count of the functional and non-functional requirements. This count is done disregarding the status of each requirement. This measure can be used in conjunction with the # initial, current, and final requirements as well as the # changes per requirement to assess the level of requirements volatility.

The mapping rule we followed in the case study was to count all atomic requirements that were in any of the states shown in Figure 5. The representation condition (properties 5 and 7) was verified during the case study, we compared the size of two SRSs and the total # requirements obtained by the two SRSs. The empirical relations were preserved by the numerical relations.

**Number of Initial, Current, and Final Requirements**

The # initial, current, and final requirements measures are equivalent to the total # requirements. These measures differ from each other in the time when the requirements are counted and the reasons for collecting them.
The initial requirements is a count of requirements at a specific point in time (for instance one week after the first draft of the SRS has been written). This includes all functional and non-functional requirements originally provided by the customer.

The final requirements is a count of all the requirements implemented by the software product. This includes all functional and non-functional requirements upon which the final software product is produced.

The current requirements is a count of the requirements collected in the current time point (day or week or similar, it depends on how often the data is collected).

By comparing the data we get from these three measures we can see how much the “total requirements” changes over time and we can do predictions of how the size of SRS will change over time. All these three measures can be used in conjunction with the number of changes per requirement to provide meaningful analysis, such as requirements stability. The properties in Figure 1 are not applied because of similarity with the previous measure. The only difference between these measures and the total requirements is in property 4. In the case of initial, current, and final requirements the mapping rule must contain a time reference (i.e. when the requirements are collected).

Status of Requirements Specifications
A requirements document (or SRS) can be in many states: it can be a draft where the requirements are only sketched or a baselined document if the requirements are validated. In general the number of states of a SRS depends on the size of the company. The bigger the company, the more probable is that the SRSs have several different states. To simplify the discussion here we assume that a SRS can have three different states: initial, draft, and final. To measure the state of a SRS we can check the states of each requirement and count the number of requirements in each state. We get an overview of the state of the SRS. This measure (status of SRS) is an indirect measure. Indirect measures are validated by following a different set of properties [12].

Status of Requirements
The status of a requirement refers to whether a single requirement has certain properties or not. An example of the possible requirements states are listed in Figure 5. The different states correspond to the phases of a requirements life cycle. As a requirement moves through the RM process, its status moves through the states listed in Figure 5. The requirements life cycle can therefore be tracked from the original proposal through to its delivery to the customer. This measure can be charted to provide details into where requirements are at a certain point in time. This can give an indication of how to allocate resources when there are still requirements in a draft stage respect to others which are in the development phase. Among the states there is a partial ordering, implying that we cannot define many empirical relations, only the binary relations “equal” and “different” can be defined.

The rule of mapping we used during the case study was to check the status of the requirements that have not been subcontracted or reused.
To verify the representation condition, we compared the states of two requirements empirically and numerically. The empirical properties were preserved. However the empirical states of the requirements are dependent by the representation of the states. It is very hard to compare requirements disregarding the representation of the states.

Regarding property 8, there is only one unit for this measure therefore the property is satisfied.

**Changes to requirements (this header is wrong, shoulnt be an attribute?)**

Another entity relevant to our discussion is change to requirements. A change to a requirement is any modification done to the semantic of the requirement.

A change may be requested either to correct a defect, to add a new feature to the system, to delete and update a feature. Each change to a requirement can be tracked.

Internal attributes for the entity changes to requirements can be size of change, rationale for change, phase of change. A change to a requirement can be considered as a new requirement and therefore can be in any of the states in Figure 5. This makes the measures “status of requirement” and “status of change to requirements” similar to each other. The measures status of change, reason of change, type of change are similar to status of requirements which has been validated previously. Therefore we will apply only property 4 to them to define the rules of mapping.

**Number of Changes per Requirement**

The # changes made can be counted, and this count can be summed for each requirement. This measure can be used to help determine requirements stability as well as to measure the impact of changes on the software process, on the budget, and on the schedule of the project. As a requirement is reviewed, all changes are recorded. This measure can be used in conjunction with other measures to chart general trends within the requirements management process.

The mapping rule we used during the case study was to count all changes to requirements requiring at least 15 minutes.

To verify the representation condition we had to prove that given two requirements A and B, if the size of change of A is bigger than the size of change of B then the # changes of A is major than the # changes of B. However we had problems to verify the representation condition during the case study, because 1) the changes to requirements were counted but not documented; 2) it is difficult to compare the size of change because the concept of “change to requirements” is not formalised.

To verify property 8 we could measure the size of change to requirements by counting the # changed words or the # changed sentences to requirements. All these units are equivalent if the measure is valid.
Status of Change to Requirements
The status of a change to requirements refers to whether a change to a requirement is in a certain status, the ones listed in Figure 5. We can track the changes to requirements from original proposal through to the test of the change. This measure can be charted to provide details into where changes to requirements are at a certain point in time. This can give an indication of how stable the requirements are.

The mapping rule used during the study was to not consider a “deleted requirement” as a change to that requirement.

Type of Change to Requirements
This measure identifies the types of changes which are common in the process. It provides insight into why requirements change. The types of change that are made to requirements include: change in delivery date, change in functionality, and other. The changes in functionality can be: correction (something was wrong), completion (something was missing), improvement (the requirement can be rewritten in a better way), and adaptation (caused for instance by new laws or new technology). These types correspond to the classifications of maintenance types. As a result, the number of types of changes can be charted to show general trends of types of changes. These types constitute the range of the measure.

The rule of the mapping we used during our study was to classify only those changes to requirements that affected the semantic (or content) of the requirements.

Reason for Change to Requirements
The reason for change identifies the types of changes which are common in the process. This measure provides insight into why requirements change. We cannot prescribe a fix set of reasons for change. The ones we used during the case study are: new government/organisation regulations, misunderstanding in original analysis, ambiguous specification, incomplete specification, wrong specification, new requirement, and other (this is also the range of the measure). These reasons can be charted to give an indication of why requirements are changed.

The rules of the mapping are the same as for the measure type of change to requirements.
<table>
<thead>
<tr>
<th>Entity</th>
<th>Internal attribute</th>
<th>External Attribute</th>
<th>Measure</th>
<th>Domain</th>
<th>Range</th>
<th>Scale</th>
<th>Empirical Relation</th>
<th>Numerical Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRS</td>
<td>requirement specification size</td>
<td>stability, change impact</td>
<td>Total # Requirements</td>
<td>SRS</td>
<td>Natural Numbers</td>
<td>ratio</td>
<td>bigger, smaller, equal to, addition, subtraction etc</td>
<td>&gt;, &lt;, =, +, -, etc.</td>
</tr>
<tr>
<td>SRS</td>
<td>requirement specification size</td>
<td>traceability, consistency, stability, volatility, change impact</td>
<td># Initial, current, final requirements</td>
<td>SRS</td>
<td>Natural Numbers</td>
<td>ratio</td>
<td>bigger, smaller, equal to, addition, subtraction etc</td>
<td>&gt;, &lt;, =, +, -, etc.</td>
</tr>
<tr>
<td>Requirements</td>
<td>size of changes per requirement</td>
<td>stability, volatility, change impact</td>
<td># Changes per requirement</td>
<td>requirements</td>
<td>Natural Numbers</td>
<td>ratio</td>
<td>bigger, smaller, equal to, addition, subtraction etc</td>
<td>&gt;, &lt;, =, +, -, etc.</td>
</tr>
<tr>
<td>Requirements Status of Requirements specifications</td>
<td>stability</td>
<td>Status of each requirement</td>
<td>(see Figure 6)</td>
<td>nominal</td>
<td>equal and different</td>
<td>= &lt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change to requirement</td>
<td>progress of change requests in life cycle</td>
<td>Status of change to requirements</td>
<td>(see Figure 6)</td>
<td>ordinal</td>
<td>equal and different</td>
<td>= &lt;&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change to requirement</td>
<td>change classification</td>
<td>traceability, consistency, stability, change impact</td>
<td>Type of change to requirements</td>
<td>changes to requirements</td>
<td>(see paragraph page 105)</td>
<td>nominal</td>
<td>equal and different</td>
<td>= &lt;&gt;</td>
</tr>
<tr>
<td>Change to requirement</td>
<td>rationale for changes</td>
<td>stability, change impact</td>
<td>Reason of change to requirements</td>
<td>changes to requirements</td>
<td>(see paragraph page 105)</td>
<td>nominal</td>
<td>equal and different</td>
<td>= &lt;&gt;</td>
</tr>
<tr>
<td>Change to requirement</td>
<td>Duration</td>
<td>Cost, effort, of change to requirements</td>
<td>Time</td>
<td>changes to requirements</td>
<td>hours</td>
<td>Interval</td>
<td>bigger, smaller, equal to, addition, subtraction etc</td>
<td>&gt;, &lt;, =, +, -, etc.</td>
</tr>
<tr>
<td>Change to requirement</td>
<td>Resource</td>
<td>cost, change impact</td>
<td>People</td>
<td>changes to requirements</td>
<td>natural number</td>
<td>Absolute</td>
<td>bigger, smaller, equal to, addition, subtraction etc</td>
<td>&gt;, &lt;, =, +, -, etc.</td>
</tr>
</tbody>
</table>
Cost of Change to Requirements

The cost of change is an indirect measure usually expressed as a function of variables like size of product, time, etc. The purpose of this measure is to give information about the actual cost spent on changing requirements and compare this information to the estimated cost. This measure can be used to assess the overall impact of requirements change on the software project. We can measure the cost in terms of time and people. The time spent on changing requirements and the number of people are direct measures and therefore can be validated through the properties in Figure 1. The internal attributes associated to requirements change are duration and resources necessary to change the requirement.

Time spent on change. When measuring time, we had several options during the case study about what rule of mapping to choose. We could measure only the time necessary to implement the change. Another alternative was to include also the time necessary to make discussions and decisions about the type of change to be implemented. We decided to include all the time necessary for discussions and decisions about the change and the time to implement the change. To verify the representation condition we had to prove that given two changes to requirements A and B, if the duration necessary to perform A is bigger than B then the time to perform A is major than the time for B. We verified empirically this property and it was satisfied. We could measure the entity change to requirements by counting the # changed words or the # changed sentences to requirements. All these units are equivalent because the measure is valid.

Number of People. We had several options regarding the mapping rules. To consider only people who perform the change or to consider also the people who discuss about the change and/or validate the change. We decided to count all the people discussing and implementing the change. To verify the representation condition we have to prove that given two changes to requirements A and B, if the amount of resources necessary to perform A is bigger than B then the # people to perform A is major than the # people of B. As before, we verified empirically this property and it was satisfied. Regarding property 8, there is only one unit for this measure (only one way of counting people).

3.5 Case study in a SE course

Overview and motivation of the study

We here depict a case study to demonstrate that the measures described previously are useful to quantify the amount of changes to requirements and to predict the cost of a change. This study was performed with the purpose of demonstrating that the intuitive cost estimations are worse than the estimations based on historical data. We followed the experiment process shown in Figure 6 [34]. However, the empirical work described here is a case study, because there were variables in the study that we could not control like the developers experience.
Informal description of the study

The study was conducted in the course Object-Oriented Software Development (OOPU) held at Umeå University. There were about twenty students in this course divided in four teams. Two teams of students collected the data during the development of their projects. The students wrote their functional requirements following a specific format [see appendix 10.1].
They measured their requirements every week and filled in forms with the results of the measurement activities on their projects. Data collection was based on use cases, since students followed a use case based approach.

The projects were run during a period of 20 weeks which were divided in two parts (iterations). The software developed in these projects was delivered to the customers. There were two customers, the teacher of the course OOPU and another customer external to the Department of Computing Science where the study was performed.

Each team received a form with a list of measures to be collected. There were two different forms, the forms differed in the amount of data to be collected. The reason for this difference is because we thought that the cost of change of a requirement depends on the type of requirements and on the type of change to requirements. The data collection was performed differently by the two teams. One team (Team B) made intuitive predictions of cost of changes to requirements during the first and second iterations. The plan for the other team (Team A) was to make intuitive predictions in the first iteration. In the second iteration their prediction should have been based on historical data. However Team A did not have changing requirements in the second iteration therefore they were not able to make predictions at all.

Risks and limitations of the study
By performing this case study, three major risks can become negative events:

Classroom projects are usually stable and well defined. There was therefore the risk that the projects under study did not have changing requirements and historical data. To minimise this risk, we met the customers to be sure that this would not happen.

The subjects could not be engaged enough to collect data. The consequence of this would be a lack of data from the participants. To minimise this risk we decided to offer an inducement to the participants of the study.

The participants could become aware of the importance of managing requirements because they know that they are participating to a case study. The students might work harder to get good effort estimations. This problem is known as the Hawthorn (or observer) effect [16]. When the project personnel become aware that their work is being monitored, their behavior will change.

Other small risks were the possibilities that the subjects use a formal model for cost predictions like COCOMO, affecting the results of the case study. There was also the possibility that the participants use requirements management tools or consult a requirements management expert. The students could have misunderstood the terminology used in the forms. This risk was low because we kept email contact with the students for all the duration of the study. The personnel experience was not investigated before the start of the study and the groups constellation was decided by the participants. This could have caused the groups to not have a uniform personnel experience in software development (the experience could have been unevenly distributed). Therefore the results of the study could have been affected by the groups experience. These risks were minimal and were managed by interviewing the students and navigating in their course site.
As mentioned in [2] there are other general risks common to all empirical studies. For instance the danger to interpret the results without attempting to understand factors in the environment that might affect data. Underestimating the resources needed to validate and analyse the data, to associate measures with wrong scale type and consequently analyse data with the wrong statistical test [7]. During the validation process, we might not be able to say how many data will never be reported. As before, these risks were minimal.

Definition of the case study
The definition of the case study determines why the case study is conducted. By applying the GQM template [1] for the goals definition we obtained the following:

- **Analyse**: Requirements Management process area in the Objekt Orienterad Program Utveckling (OOPU) course.
- **With the Purpose of**: Evaluate the impact of software measurement in prediction of cost of changes to requirements.
- **Quality focus**: Effectiveness of the use of historical data in predictions of cost of changes to requirements compared to intuitive predictions.
- **Perspective**: Academy.
- **Context**: The study is conducted in an object-oriented software development course. The context of the study will be described below in more details.

Planning of the case study
The planning of a case study determines how the study is conducted. The design of the case study is decided in this phase.

We could identify a causal relationship in our case study idea. The use of historical data for predictions of cost of changes to requirements has the effect of increased precision in cost predictions.

For accuracy reasons, our plan has been to proceed concurrently between data collection and data validation. The validation consisted in checking the forms for correctness, consistency, completeness, omissions, and miscategorisation. In case where the checks revealed problems, participants were interviewed. As suggested by [2], such validity checks resulted in corrections to the data that could not be made in a later stage due to the nature of human memory.

The context of the study was a course (OOPU) focused on object-oriented approaches. Methods, languages, and tools that support these approaches were discussed and applied. The work was documented step by step in workbooks and project reports. The course was project oriented, the projects were small-medium size (the total effort consisted in ten thousands LOC developed by 12 persons in a month). The projects under study were run in a off-line environment (non industrial software development) (see [jubo jcse for details about the course and the process used in the course].
The subjects of the case study were students of the 3rd to 5th year of a Computing Science education program held at Umeå University, Sweden. The subjects were attending the OOPU course described above. Some of the students had previous background knowledge in software development acquired outside the university. Therefore, the participants were semi-professional and non-professional developers. The students were organised into 4 teams of about six members. Each team worked through all phases of the software development process, from project proposal to the implementation and delivery of a working prototype. According to our plan, all the four groups should have participated to the study. However only two groups collected data.

The developed applications were "Inredaren" and "UmUportal". The first one is a floor planning system that provides the user with a window-based user interface. It has two main functions, floor drawing and floor furnishing. UmUportal is a personal portal for Umeå university.

The students used Rational Rose to support analysis and design. The cost of a change was calculated through all the activities and affected by the tool (see [9] for details on the course)

The study was specific since it was focused on managing requirements. It addressed a real problem i.e. the difference in predicting cost of changes to requirements with or without historical data.

The objects under study were Software requirements specifications (SRS), and the software process used.

The null hypothesis H0, the one to be rejected, was the following: the cost predictions made by using historical data are at most as good as the intuitive predictions. The alternative hypothesis H1 was the following: the cost predictions made by using historical data are better than intuitive predictions.

One independent variable was the personnel experience. The project was another independent variable, with values "Inredaren" and "UmUportal". There was one factor, cost predictions and two treatments (the values of the factor): intuitive cost predictions, and controlled cost predictions. The dependent variable was "precision of cost predictions".

All the students participating to the OOPU course were selected. The students were divided into 4 groups. Each group had at least one team manager and one requirements engineer. Each student in the group could assume different roles during the project.

The design type chosen was one factor with two treatments. The design principle followed was balancing i.e. to assign the treatments so that each treatment had equal number of subjects. In our case we planned that two groups of students should work on the development of a student portal and the other two groups should develop a Floor Planning System. The forms were assigned such that each project could be evaluated by two different forms. Two teams should have performed intuitive cost predictions, two teams of students should have performed cost prediction using historical data.

The kind of statistic chosen for our measures: # initial, current, final requirements, # changes to requirements, time, and for # people, was a parametric statistic because as suggested by [22] these measures reach at least the interval level.
Non parametric statistic tests make less stringent assumptions. For the measures type of change to requirement, reason of change to requirement, status of requirement and status of change to requirements we choose non parametric tests because these measures reach the nominal-ordinal level.

The instruments are usually of three types: objects, guidelines, and measurement instruments. The only instrument used during the study was e-mail forms in plain text. The reason for choosing plain text forms was because of simplicity and flexibility in the answers. Other ways of collecting data have been considered like excel-forms and html-forms available on line but these were discarded because they are more complex and less flexible. Team A (the controlled group) used a form containing a detailed list of measures to collect and predicted the cost of change to requirements based on the data collected and historical data. Team B used a form containing a simplified list of measures and made intuitive estimation of cost of changes to requirements. The forms are shown in the appendix 10.1 and 10.2. The forms have been informally reviewed.

Case study operation
The preparation of the subjects was done during a lecture where we explained the background of the research, the case study goals, and we showed the forms to be filled in.

During the execution of the case study, the main concern was to ensure that the study was conducted according to the plans. The case study was executed during a period of four months (from September 2002 to January 2003). The costs were limited to a small inducement.

The measures used during the study are the ones shown in Table 2. We measured requirements with a use cases granularity.

Analysis and interpretation
The study did not succeed as expected for two main reasons. First: among the four groups performing a project in the OOPU course, only two teams participated to the case study. Even an inducement did not convince the other two teams to collect data. Secondly: the projects did not have many changing requirements. The original goal was that the controlled group should make cost estimation in the second iteration by using historical data. We expected meetings between the customer and the developers to validate the requirements, and as a consequence, a series of requests of change from the customer. Sadly, the controlled group had little contact with the customer to validate their requirements and this affected the results of the study.

During the data collection process there was e-mail contact with the students about how to collect time and people measures and which “changes to use cases” were relevant to the study. We decided not to collect data related to changes to use cases whose time to perform the change was less than fifteen minutes. There has also been some misunderstanding in how the data should be collected. In fact the changes to use cases were collected in two different
Another discussion with the students was related to the use cases states. Some of the use cases were "reused", others were "subcontracted". We did not foresee these states, therefore they were not present in the forms provided to the students.

The descriptive statistics help us to understand and interpret data informally. This is the first step to be applied after collecting the data. Descriptive statistic may be used to describe and graphically present interesting aspects of a data set [34].

By comparing data collected by the two teams (Figure 1, Figure 2, Figure 6 and Figure 7), we can observe that the graphs are similar. Figure 1 describes the measure # use cases per week for both teams, connected to the stability of requirements. Team A has a period during the first 5 weeks of the project when the requirements are changed. Afterwards, Team A's requirements are very stable. The same situation can be observed in Figure 6 (Team A use cases per week). Team A has some changes to requirements only during week 41. During the second iteration, the # changes is equal to zero. The same situation can be observed in Figure 7 which describe the measure Team B # use cases per week.

Observing Team B use cases in Figure 1, we can notice very well the two iterations in which the project is divided. The requirements are changing during the first two weeks of the first iteration and between the two iterations. The same behaviour can be observed in Figure 2. The # changes per requirements per Team B is higher during the beginning of the first iteration and it decreases to zero. Between the first and second iterations there are other changes. Similar situation can be observed in Figure 7 which describes the measure Team B # use cases per week (The mode is equal to zero for both groups).

In Figure 3, 4, and 5 we can observe the difference between the actual and expected cost (time, people, effort) for Team B. There is no real difference in estimations between the first and second iteration. The estimated values are quite similar to the actual values.

Finally, in Figure 8 and 9 we can observe the status of the use cases for Team A and Team B. Among the status proposed, Team A introduced two states "deleted" and "reused", while they never used the states "reviewed", and "tested". Similarly, Team B used two non suggested states, "deleted" (same as for Team A) and "subcontracted".

3.6 Discussion and conclusion

In the previous sections we have validated requirements management process measures (shown in Table 2). After that we have used these measures in a case study performed in a university environment. The goal of the study was to demonstrate that estimations of cost predictions done by means of historical data are better than intuitive predictions.

We encountered some difficulties in performing the validation of the measures. For example it was sometimes difficult to find internal attributes and to distinguish between internal attributes and direct measures (as pointed out by [wohlin]).
Difficult to verify the representation condition for status measures and for other measures. In the case of changes to requirements, it was difficult to compare the size of change made to two requirements because we did not have a formal definition for size of change. Furthermore, the list of attributes and empirical relations for the attributes might not be a complete list.

As we have pointed out earlier, the study did not succeed for several reasons. However we have learned some lessons which are listed below.

- When doing measurement it is very important to define entities carefully. For instance in our case it was difficult to decide what a requirement is, what granularity should a requirement have when we measure it, what format should the requirement have. All these issues must be taken into account when we measure, otherwise it is difficult to compare the results of the measurement activities.
- An inducement is not enough to engage students to participate in a case study. More effort should be spent in committing the participants and obtaining their consensus.
- More effort should be spent in pushing students and customers to discuss about their requirements. The subjects belonging to the controlled group and their customer had few discussions about requirements. We expected validation of the requirements from the students side and requests of change from the customer side.
- No restrictions should be put in data collection especially in student projects which are more stable than real on-line projects. The decision not to collect data about small modification of use cases was probably a mistake because by doing so the number of changes decreased considerably. However the time spent to report the data should not overcome the time spent in implementing the change. The best would be to use an automatic data collection tool so that the time to register data does not overcome the actual time of work.
- With regards to the measures definition, new values for the “status of requirements” measure were introduced (cancelled, reused, and subcontracted). These new values made the measurement scale of the status of requirements to become nominal instead of being ordinal. This can affect the kind of statistic we choose. For other nominal and ordinal measures new values were introduced and some values were not used.
- The measure # changes per requirements was collected in different ways by the two groups. The reasons for all these misunderstandings can be a non-strict definition of the mapping rules and a consequent confusion of the participants collecting the data.

Although we were not able to complete our study successfully, this work can be useful as an example of how to perform a case study in a university environment and what risks should be taken into account. Furthermore, a lot of research is done to define properties of valid software measures but, to our knowledge, few show examples of application of these properties to the measures. Our work tries to fill this gap.
3.7 Future work

As future work we plan to improve the present research in several ways. As we have stated earlier, each measure should be part of a prediction system, i.e. for each measure we should demonstrate that the external attribute is a function of the internal one. Table 2 can be the starting point for creating prediction models.

Some concepts should be studied more deeply. For instance, is it reasonable to speak about the size of requirements? Or are we considering the size of the software/project instead? Some definitions need to be formalised like change to requirement. The study of the entities and the relationships among the entities is another topic that can be investigated. We can also validate what is left from the 38 measures defined in [14]. These measures need to be tailored to the specific organisation.

We plan to perform an empirical study in a medium-size company in Sweden. We will analyse historical projects at this company and based on this analysis we will design a suitable requirements management process at this company. After that we will compare the historical data with the actual data obtained following the new requirements management process. Finally, we will create a baseline which can be used for future projects as a reference for comparisons.

Another possible empirical study we would like to perform is to demonstrate that the size of software systems is proportional to the size of their requirements, in other words: if $S$ and $R$ are two requirements and $\text{size}(R_1) < \text{size}(R_2)$ the software systems $S_1$ and $S_2$ generated by these requirements are such that $\text{size}(S_1) < \text{size}(S_2)$.

3.8 Acknowledgement

Thanks to the students of the OOPU course that have participated to the study for their help by collecting data. In particular I would like to thank Magnus Andersson, Marcus Bergner, Martin Englund, Markus Holmberg, Mattias Nordin, Fredrik Åslund, Jan Svensson, Olof Burman, Josef Israelsson, Hans Olofsson, and Rickard Melkersson.

3.9 References


Appendix A.

Form 1

The following form was distributed by email to the controlled group named Team A.

- Project group name: ________
- Week: _______
- Number of current use cases (right now): ________

Please identify the use cases with a number and for each use case write:

- Use case number: ________
• Use case status:____________
  where status can be: started, documented, reviewed, implemented, tested, delivered, other
  (specify).
• Number of changes to use case:_______
  Please identify the changes with a number and for each change to use cases write:
• Change number:_______________
• Status of change:______________
  where status can be: proposed, approved, rejected, designed, implemented, tested, other
  (specify).
• Type of change:_______________
  where type of change can be: change in delivery date, change in functionality: correction
  (something was wrong), completion (something was missing), improvement (the use case can
  be rewritten in a better way), adaptation (caused for instance by new laws or new technology),
  other (specify).
• Reason of change:______________
  where reason can be: new government/organisation regulations, misunderstanding in
  original analysis, ambiguous specification, incomplete specification, wrong specification,
  new (customer) requirement, other (specify).

  Estimated cost of change, where estimated cost of change is
• total time the team plan to work on the change in hour:__
• number of people who will work on the change:______

  Actual cost of change, where actual cost of change is
• total time the team has spent on working on the change in hour:__________
• number of people who have worked on the change:____
  Other comments:____________

Form 2
The following form was distributed by email to the controlled group named Team B.
• Project group name:________
• Week:______
• Number of current use cases (right now): ___________
  Please identify the use cases with a number and for each use case write:
• Use case number:___________
• Use case status:_____________
  where status can be: started, documented, reviewed, implemented, tested, delivered, other
  (specify).
• Number of changes to use case:_______
  Please identify the changes with a number and for each change to use cases write:
• Change number:______________
Estimated cost of change, where estimated cost of change is
• total time the team plan to work on the change in hour: ___
• number of people who will work on the change: _______

Actual cost of change, where actual cost of change is
• total time the team has spent on working on the change in hour: _______
• number of people who have worked on the change: _______
Other comments: ______________

Use case template
During the case study, the use cases were written by the participants according to the template which follows.

Table 2: Use Case Template

<table>
<thead>
<tr>
<th>Fields</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case</td>
<td>Memorable name (active verb-goal phrase) and unique ID</td>
</tr>
<tr>
<td>Description</td>
<td>Short description of the goal to be achieved</td>
</tr>
<tr>
<td>Actors</td>
<td>List of actors involved in the use case</td>
</tr>
<tr>
<td>Preconditions/assumptions</td>
<td>Conditions that must be satisfied to trigger the use case</td>
</tr>
<tr>
<td>Main success scenario</td>
<td>Sequence of interactions between the actor(s) and the system</td>
</tr>
<tr>
<td>Postcondition</td>
<td>Conditions that are satisfied after successful completion</td>
</tr>
<tr>
<td>Non-functional requirements</td>
<td>Non-functional requirements related to the use cases (may be described in detail elsewhere)</td>
</tr>
<tr>
<td>Open issues</td>
<td>List of issues that have to be solved</td>
</tr>
<tr>
<td>History</td>
<td>Modification history</td>
</tr>
</tbody>
</table>

Descriptive statistic
FIGURE 8. Ikaros and Forc number of use cases per week

FIGURE 8. Ikaros and forc number of changes per week
FIGURE 9. Ikaros time spent on changing use cases

FIGURE 10. Ikaros number of people who changed use cases
FIGURE 11. Ikaros effort in changing use cases (people/time)

FIGURE 12. Fore number of use cases stable, changed and deleted
FIGURE 13. Ikaros number of use cases changed, stable and deleted

FIGURE 14. Forc use cases status
FIGURE 15. Ikaros use cases status
Preliminary results of two academic case studies on cost estimation of changes to requirements

Annabella Loconsole, Jürgen Börstler

Abstract

Requirements management measures can help us to control changing software requirements and estimate the costs of changing requirements. This paper describes two small case studies, performed in the context of a team-project based software development course. In the first study, we compared intuitive cost estimations of changes to requirements to estimations based on historical data. In the second one, we studied whether even simple tools to support impact analysis affect the accuracy of cost estimations. Although the data we collected in these studies is not suitable for statistical analysis, we can present some interesting results and lessons learned. Our results suggest that estimations based on an impact analysis checklist are better than the intuitive estimations obtained in study one. However, study two is not yet completed therefore we cannot draw further conclusions.
4.1 Introduction

Carefully developed software requirements are a key issue for project success [21]. The cost of correcting an error after system delivery is orders of magnitude higher than the cost of correcting a similar error during the requirements analysis phase [17]. Since requirements change during software development, it is important to control those changes to be able anticipate and respond to change requests [19]. Requirements development is a learning process rather than a gathering process. Therefore, it is naïve to believe that we can specify exactly what a customer wants at the beginning of a project. The best we can do is to carefully monitor and control all requirements throughout the software life cycle.

Software measurement can help us in providing guidance to the requirements management activities by quantifying changes to requirements and in predicting the costs related to changes. Numerous software measures for the requirements management activities have been proposed in the literature (see [4], [18], [10], [20], [8]). However, few empirical studies [1], [11], [12], have been performed to demonstrate the effectiveness of these measures. In our previous work [13] we analysed the key practices defined in the Requirements Management Key Process Area (KPA) of the SW-CMM [16]. By means of the Goal Question Metrics (GQM) paradigm [2] we defined a total of 38 measures.

In this paper, we will present partial results of an ongoing case study and compare it with a previous one. The motivations to perform these studies and compare them are manifold. Our first goal is to investigate whether cost estimation of changes to requirements performed using historical data are better than intuitive cost estimations. The second goal is to investigate whether cost estimation of changes to requirements based on detailed impact analysis are better than intuitive cost estimations. Another goal is to compare the case studies and show the methodological improvements accomplished by using the lessons learned from the first case study. Furthermore, we want to contribute to the lack of empiricism in the area of requirement management measures.

Study one did not reveal sufficient data to draw statistically significant conclusions because the teams did not have changing requirements in the second iteration therefore they were not able to make predictions. At the time of writing, study two is not yet completed therefore we have only some preliminary results. Among the results, we can say that one of the teams who collected data based on impact analysis checklist had good estimations. However, the data collected is still too little to be able to deduct some conclusions.

We roughly followed the experiment process proposed by [22]. However, the empirical work described here are case studies, because we have had little control over the variables involved in the studies.

The remainder of this paper is organised as follows: section 2 presents an overview of the studies, in section 3 we describe the context of the studies. Section 5 contains the risks of the studies. Hypothesis, plans and measures are described in section 6. Sections 7 and 8 describe the results for study one and two respectively. In section 9 we compare the two studies. Finally some conclusions and future work are presented in section 10.
4.2 Case studies overview

Both studies were conducted with student teams in the context of the course Object-Oriented Software Development (OOSD) held at Umeå University. In fall 2002 there were twenty students in this course divided into four teams. According to our plans, all the four teams were required to participate to the study. However, only two teams collected data on a regular basis. In fall 2003 there were twenty-six students divided into five groups. All teams are participating in the study.

The students measured their requirements every week and filled in forms with the results of the measurement activities on their projects. In both studies, the teams described their functional requirements following a specific format [14]. Requirements were measured regularly and the results submitted weekly using predefined forms. The projects had a schedule of twenty weeks. The software development process followed in the course was a two iterations process where each iteration was eight weeks long. Each team received a form with a list of measures to be collected.

In study one, only two teams performed the data collection on a regular basis (team A and team B). Team A made intuitive predictions of costs of changes to requirements during the first and second iterations. The plan for team B was to make intuitive predictions in the first iteration. In the second iteration, their predictions should be based on historical data, collected during the first iteration. However, team B did not record any changes to requirements in the second iteration and therefore did not provide the required predictions.

In study two, we decided to not have a distinction between iterations, since twenty weeks was too short a period to collect sensible historical project data. Two teams made intuitive cost estimations while the other three teams based their estimations on a checklist-based impact analysis. At the time of writing, one team is collecting data on a regular and precise way, the data collected by the other teams are not yet complete.

4.3 Definition of the case studies

By applying the GQM template [2] for the goals definition, we obtained the following for study one:

Analyse: Requirements Management process area in the OOSD course.

With the Purpose of: Evaluate the impact of software measurement in prediction of cost of changes to requirements.

Quality focus: Effectiveness of the use of historical data in predictions of cost of changes to requirements compared to intuitive predictions.

Perspective: Academy.

Context: The study is conducted in an object-oriented software development course. The context of the study will be described below in more details.

Study two differs from the first only in the purpose and quality focus:
Analyse: Requirements Management process area in the OOSD course.

Purpose of: evaluating the cost model based on detailed impact analysis of cost of changes to requirements.

Quality focus: accuracy of the cost model.

Perspective: Academy.

Context: The study is conducted in team-project object-oriented software development course.

4.4 Case studies context and environment

The context of the studies was a team-project software development course (OOSD), focusing on object-oriented approaches. Methods, languages, and tools that support these approaches were discussed and applied. Projects had a schedule of 20 calendar weeks and students are expected to spend 20h per week on this course on average. Given six students on an average team this results in an effort of about 12-13 person months. Projects span all phases of software development, from initial customer contact to the delivery of a product. Most projects have external customers. All project work has to be documented in detail by means of deliverables, presentations, prototypes, weekly reports, diaries and an on-line project workbook. Details about the course and the development process used can be found in [7].

The subjects of the case studies were 3rd to 5th year Computing Science students. The subjects were attending the OOSD course, held at Umeå University, Sweden, described above. All students had good knowledge in programming and had taken a basic software engineering course before entering OOSD. Some of the students also had some experience from software development outside the university. Therefore, the participants can be seen as semi-professional developers. In study one there were four teams of five members. The developed applications were "Inredaren" and "UmUportal". The first one is a floor planning system that provides the user with a window-based user interface. It has two main functions, floor drawing and floor furnishing. The application was ordered by a teacher of the computing science department (internal customer). UmUportal is a personal portal for Umeå University. The application was ordered by an external customer, the IT chief of Umeå University. In study two, the students were organised into five teams of five-six members each. The developed applications were an "editor for XML metadata" and a "Course Pre-requisites Checking System". The first application includes a textual and graphical (WYSIWYG) editor for XML-based forms, support for undo, and support for internationalised forms. The second application is an on-line, web-based system for course registration and simple curriculum management. In both cases, the customers were external to the computing science department. The first customer was a high-tech company active in the areas of digital media management, networking and high performance computing in Umeå. The customer of the second application was the faculty of teacher education in Umeå.

The students were required to use specific support analysis and design tools (Rational Rose in study one and Together ControlCenter in study two). The cost of a requirements change was calculated throughout all software development activities, i.e. from analysis to integration
into the prototype, including the update of all affected documents. footnote: Approximately 20% of the total time available is spent on lectures and exercises, the learning of new methods, languages, and tools and course administration. A person month amounts to about 152 hours of work, according to COCOMO.

The studies were "specific" since they were focused on managing requirements. They both addressed a real problem, i.e. the difference between intuitive and historical based predictions in study one and the difference between intuitive predictions and predictions based on detailed impact analysis of cost of changes to requirements in study two. The objects under study were Software Requirements specifications (SRS), and the software process used.

4.5 Limitations of the case studies
By performing these case studies, we identified three major risks that could become negative events:

- Classroom projects are usually stable and well defined. There was therefore the risk that the projects under study did not have changing requirements and historical data. To minimise this risk, we contacted the customers and agreed with them to change the requirements of the projects. This was especially true in study two.
- The subjects might not be motivated to participate in the case studies. The consequence of this would be a lack of data from the participants. To minimise this risk we decided to offer an inducement to the subjects in study one, while in study two we told the students that the data collection was a requirement to pass the course.
- By participating to the case studies, the subjects could become aware of the importance of managing requirements. The students might work harder to get good cost estimations. This problem is known as the Hawthorn (or observer) effect [15]. When the project personnel become aware that their work is being monitored, their behaviour will change. In both studies, this risk was lost since the course require a considerable amount of work therefore we did not expect the students to work more only to get better estimations.

Other minor risks were the possibilities that: 1) the subjects could use a formal model for cost predictions like COCOMO, affecting the results of the case study; 2) the participants could adopt requirements management tools or consult a requirements management expert; 3) the students could have misunderstood the terminology used in the forms. This risk was low because we kept e-mail contact with the students during the studies. The personnel experience was not investigated before the start of the studies and the participants decided the team’s constellation. Therefore the results of the studies could be affected by the team’s experience. We thought that these risks were minimal and were managed by interviewing the subjects and navigating in their course web site.

There were other general risks common to all empirical studies [3], for instance the danger to interpret the results without attempting to understand factors in the environment that might affect data. Underestimating the resources needed to validate and analyse the data, to associate measures with wrong scale type and consequently analyse data with the wrong statistical test [6] are other risks. During the validation process, another risk was to not know
the amount of data that will never be reported. However, these risks were minimal. Unfortunately, the first two risks described above became partially true in study one resulting in too little data collected to be able to draw statistical conclusions.

4.6 Hypotheses and plans
In study one, we assumed that intuitive cost estimations are at least as good as estimations based on historical data (the null hypothesis H0). For the alternative hypothesis H1 we assumed that intuitive cost estimations are worse than estimations based on historical data. The null and alternative hypotheses in study two are similar to the first study, the difference is in the estimations based on an impact analysis checklist rather than historical data.

The independent variables were the personnel experience and the project. There was one factor, cost predictions and two treatments: intuitive cost predictions, and controlled cost predictions. The dependent variable was "precision of cost predictions". All the students participating to the OOSD course were selected. Each team had at least one team manager and one requirements engineer. Each student in the team could assume different roles during the project. The design type chosen was one factor with two treatments.

The design principle followed was balancing i.e. to assign the treatments so that each treatment had equal number of subjects. In study one, two teams of students developed a student portal and two teams developed a Floor Planning System. We assigned the forms such that two different forms could evaluate each project. According to our plans, two teams should have performed intuitive cost predictions, and two teams should have performed cost prediction based on historical data. However, as written above, only two teams performed the data collection. In study two we had five groups therefore we assigned the forms such that at least two different forms could evaluate each project.

The kind of statistic chosen for our measures: # requirements, # changes to requirements, and time, was a parametric statistic because as suggested by [5] these measures reach at least the interval level. The instruments are usually of three types: objects, guidelines, and measurement instruments. The only instrument used during the study was forms contained in plain text e-mails. This choice was accomplished because of simplicity and flexibility in the answers.

Description of measures
In both studies, the measures collected in relation to the hypothesis described above are: # requirements, # changes per requirement, and cost of changing requirements. Other measures have been collected to have a general overview of how the requirements are managed in students' projects and to perform an internal validation of those measures [14].
Figure 1: A simplified requirements life cycle

The number of requirements is obtained by counting the functional and non-functional requirements. This counting is done disregarding the status of each requirement (a list of possible requirements states is shown in figure 1). The measure of requirements can be used in conjunction with the changes per requirement to assess the level of requirements volatility. The measurement rule followed in the case studies was to count all requirements that were in any of the states shown in figure 1. We excluded from the count only the deleted requirements.

The changes made to requirements can be used to help determine requirements stability as well as to measure the impact of changes on the software process, on the budget, and on the schedule of the project. As a requirement is reviewed, all changes are recorded. The measure of changes made to requirements can be used in conjunction with other measures to chart general trends within the requirements management process. The measure of changes to requirements includes any change to a requirement that affects the development of the requirements. A requirement deletion is not considered to be a change to a requirement and readability improvements to the requirements specifications are not considered changes unless they affect the development of the requirements. The measurement rule used during the
The cost of change is an indirect measure usually expressed as a function of variables like size of product, resources etc. The purpose of this measure is to provide information about the actual cost for changing requirements and compare this information to the estimated cost. This measure can be used to assess the overall impact of requirements change on the software project. We measured the cost in terms of resources used (the time spent on performing the change). This is because in software projects usually the staff cost dominates the overall project cost [9]. In study one, we did not define directly how to calculate the resources. Only after some discussions with the students we decided to consider the resources necessary for analysis of the change and for implementation of the change. If a change request affected many requirements, the cost of change to requirements was calculated in average for each requirement. In study two we considered the resources necessary to implement the change and not the resources for analysing the impact of change.

To test our hypothesis, each team attending the Object Oriented Software Development course collected data for the measures described above and other information useful to document the changes to requirements. Data was collected for the measures shown in Table 1.

### 4.7 Results of case study one

The subjects were prepared during a lecture where we explained the background of the research, the case study goals, and we showed the forms to be filled in. During the execution of the study (from September 2002 to January 2003), the main concern was to ensure that the study was conducted according to the plans. We had e-mail contact with the students about how to collect the measures related to time and which "changes to requirements" were relevant to the study. There were some misunderstandings in how the data should be collected.

<table>
<thead>
<tr>
<th>Entity</th>
<th>External Attribute</th>
<th>Measure</th>
<th>Domain</th>
<th>Range</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Requirement Specification</td>
<td>Stability, Change impact</td>
<td># Requirements SRS</td>
<td>Natural Numbers</td>
<td>Ratio</td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td>Stability, volatility, change impact</td>
<td># Changes per requirement</td>
<td>Requirements</td>
<td>Natural Numbers</td>
<td>Ratio</td>
</tr>
<tr>
<td>Change to requirement</td>
<td>Cost, effort of change to requirements</td>
<td>Time</td>
<td>Changes to requirements</td>
<td>Minutes</td>
<td>Interval</td>
</tr>
</tbody>
</table>
For instance, one team counted the # changes to requirements while the other team counted the # change requests. One team considered a deleted requirement as a change to a requirement while the other team did not consider it in the same way.

**TABLE 2: Summary of data collected in study one**

<table>
<thead>
<tr>
<th>Measures</th>
<th>First Iteration</th>
<th>Second iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Team A</td>
<td>Team B</td>
</tr>
<tr>
<td># Requirements</td>
<td>29</td>
<td>12</td>
</tr>
<tr>
<td># Changes</td>
<td>26</td>
<td>9</td>
</tr>
<tr>
<td>Estimated Time per change</td>
<td>24</td>
<td>17</td>
</tr>
<tr>
<td>Actual Time per change</td>
<td>26</td>
<td>40</td>
</tr>
</tbody>
</table>

A summary of the data collected is shown in table 2. The measurement unit for time is minutes. The estimated and actual time in the cells are sums of data collected every week divided by the # changes. In average, the time spent for each change was 40 minutes for team B and 26 minutes for team A. The reasons for changing use cases were in general for purposes of correction and improvement.

The requirements were changed mainly during the first iteration of the project. Afterwards, the requirements were very stable for both teams. Team B had some changes to use cases only during week 41. Team A changed use cases during the first two weeks of the first iteration and between the two iterations.

In table 2 we can observe the difference between the actual and expected time. Team B’s estimation was approximately less than half of the actual time necessary to perform the change. During the second iteration, team A’s estimation was one third of the actual time necessary to perform the change.

**Lessons learned in study one**

As pointed out earlier, the study did not succeed as expected for two main reasons. First: among the four teams performing a project in the OOSD course, only two teams participated in the case study. The other two teams decided to not collect data. Secondly: the requirements were not as volatile as expected. According to the original plan, the controlled team should use historical data to estimate the cost of change in the second iteration. We expected meetings to take place between the customer and the developers in order to validate the requirements, and as a consequence, a series of requests of change from the customer. Sadly, the controlled team had little contact with the customer to validate their requirements and this affected the results of the study. However we have learned some lessons, which are listed below.

- An inducement is not enough to engage students to participate in a case study. More effort should be spent in committing the participants and obtaining their consensus.
• More effort should be spent in pushing students and customers to discuss their requirements. The subjects belonging to the controlled team and their customer had few discussions about requirements. We expected validation of the requirements from the student side and requests of change from the customer side.

• When doing measurement it is very important to carefully define the measurement rules. The measure # changes per requirements was collected in different ways by the two teams. The reasons for this misunderstanding can be a non-strict definition of the mapping rules and a consequent confusion of the participants collecting the data. Strict definitions of measures and measurement rules are crucial when we perform empirical studies, otherwise it is difficult to evaluate the results of the measurement activities.

As we can see in table 2, the data available are only intuitive estimations and these are not very precise.

4.8 Partial results of case study two

As for study one, we prepared the subjects during a lecture in the beginning of the course. We decided to not show the details of the forms to be filled in during this lecture. Instead we provided the subjects with a document containing detailed instructions of how the data should be collected. The instructions contained the definitions of the measures, examples of how to collect the measures and the forms to be filled in. Two different instruction documents were designed, one for the teams whose estimations were intuitive and other for the teams making estimations based on an impact analysis checklist. Among the examples, we described how to count time. The measurement of time spent in performing the change was the sum of the single times spent by each developer in the team. This measure should include the time necessary for a change to requirement to be developed (postponed, approved, implemented etc.).

At the time of writing, study two is still ongoing. The measures collected during study two are the same as for study one (shown in table 1). Data has been collected from week 40 to week 46 i.e. the first seven weeks of the project schedule.

<table>
<thead>
<tr>
<th>Measures</th>
<th>Team 1</th>
<th>Team 2</th>
<th>Team 3</th>
<th>Team 4</th>
<th>Team 5</th>
</tr>
</thead>
<tbody>
<tr>
<td># Requirements</td>
<td>29</td>
<td>9</td>
<td>17</td>
<td>28</td>
<td>25</td>
</tr>
<tr>
<td># Changes</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Estimated time per change</td>
<td>165</td>
<td>--</td>
<td>4800</td>
<td>--</td>
<td>35</td>
</tr>
<tr>
<td>Actual Time per change</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>25</td>
</tr>
</tbody>
</table>

As we can observe in table 3, once again we had troubles in collecting data. The "--" in the table cells stand for "not submitted". The time is expressed in minutes. Only one team (team 5) submitted data on a regular base. The other teams missed the weekly deadlines several times, furthermore their data is incomplete. The reasons of this can be manifold. The course
requires large amount of documentation and the weekly forms for the data collections could be a further burden that the subjects try to avoid. Another reason could be due to the understanding of the subjects that the data collection is not a requirement to succeed the course. However, the study is still ongoing therefore we are still in time to make adjustments to the forms and the study. Observing data for team 5 we can notice that the estimations are quite accurate. This can be due to the way team 5 is collecting data i.e. using an impact analysis checklist. However, team 5 had only three changes therefore data is not enough to draw conclusions.

4.9 Comparison of the two case studies

In performing study two we accomplished some of the lessons learned in study one. Among the improvements obtained, all the teams participated to the study while in study one only two teams participated. Although the data collection was not a requirement to succeed the course, in study two we made the students to believe that. Another lesson learned in study one was to define the measures more carefully therefore we distributed a document with the measures definition. In fact, we did not receive questions about how to collect time necessary to change requirements. Furthermore, we contacted the customers and we invited them to ask for changes. The changes to requirements in both studies were due to customer’s change requests. However, we have probably done new mistakes. As we can see in table 3, the data collected was incomplete, imprecise and not delivered regularly. We suspect that the reason of this new problem was due to the large amount of data we were asking for.

A summary of the differences between the two studies can be seen in table 4. A full comparison of the results of the two studies cannot be performed because study two is ongoing. However a partial comparison can be observed in table 5. Team 5 has performed estimations following an impact analysis checklist while teams A and B have performed intuitive estimations. The estimations of team 5 are quite accurate compared to the estimations of team B in

<table>
<thead>
<tr>
<th></th>
<th>New study</th>
<th>Old study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intuitive data compared with</td>
<td>Impact analysis</td>
<td>Historical data</td>
</tr>
<tr>
<td>Measures definition</td>
<td>Yes</td>
<td>Partially</td>
</tr>
<tr>
<td>Instructions provided</td>
<td>Yes</td>
<td>Partially</td>
</tr>
<tr>
<td># teams</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td># students</td>
<td>26</td>
<td>20</td>
</tr>
<tr>
<td># teams who collected data</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Both forms shown to all students</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Well collected data</td>
<td>No (not yet)</td>
<td>Yes</td>
</tr>
<tr>
<td>Applications</td>
<td>Editor for XML metadata, Course Pre-requisites Checking System</td>
<td>Inredaren, UmUportal</td>
</tr>
</tbody>
</table>
the first iteration and team A in the second iteration. This can be due to the use of the checklist. However, the estimations are not good compared to Team A in the first iteration. Furthermore the data collected is still too little to be able to draw conclusions.

4.10 Conclusion and future work

In the previous sections we have described two case studies performed in a university environment. The goals of the studies were to compare intuitive cost estimations of changes to requirements with estimations based on historical data (study one) and estimations based on detailed impact analysis (study two). In performing study two we reused the lessons learned from study one as an example of what can go wrong and what risks should be taken into account.

At the time of writing, no conclusions can be drawn because study two is still ongoing and because the data collected is not complete. However as preliminary results, one team in case study two made quite accurate estimations of cost of changes to requirements performed by following an impact analysis checklist.

Currently, we are performing an empirical study in a medium-size company in Sweden. We are analysing historical projects at this company and based on this analysis we will design a suitable requirements management process for this company. After that we will compare the historical data with the actual data obtained following the new requirements management process. Finally, we will create a baseline which can be used for future projects as a reference for comparisons.

For the next year, we plan to perform another case study. We intend to compare estimations based on COCOMOII model and estimation based on impact analysis checklist. The study will be integrated into the course. Students will not know that there is a specific study. Course organisation material will be adapted to ensure that sufficient data is submitted. Another possible empirical study we would like to perform is to investigate if the size of software systems is proportional to the size of their requirements. In other words, we wonder if R1 and R2 are two requirements and size(R1) < size(R2), the software systems S1 and S2 generated by these requirements are such that size(S1) < size(S2).

<table>
<thead>
<tr>
<th>Measures</th>
<th>New Study</th>
<th>Old Study</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Team 5</td>
<td>Team A</td>
</tr>
<tr>
<td># Changes</td>
<td>3</td>
<td>26</td>
</tr>
<tr>
<td>Estimated Time per change</td>
<td>35</td>
<td>24</td>
</tr>
<tr>
<td>Actual Time per change</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>Error</td>
<td>40%</td>
<td>10%</td>
</tr>
</tbody>
</table>
4.11 Acknowledgement

Thanks to the students of the OOSD course Fall 2002 and Fall 2003 that have participated in the studies for their help by collecting data.

4.12 References


An Industrial case study on requirements volatility measures

Annabella Loconsole, Jürgen Börstler

Abstract

Requirements volatility is an important risk factor for software projects. Software measures can help in quantifying and predicting this risk. In this paper, we present an industrial case study that investigated measures of volatility for a medium size software project. The goal of the study was twofold: 1) to empirically validate a set of measures associated with the volatility of use case models (UCM); 2) to investigate the correlation between subjective and objective volatility. Measurement data was collected in retrospect for all use case models of the software project. In addition, we determined subjective volatility by interviewing stakeholders of the project. Our data analysis showed a high correlation between our measures of size of UCM and total number of changes, indicating that the measures of size of UCMs are good indicators of requirements volatility. No correlations was found between subjective and objective volatility. These results suggest that project managers at this company should measure their projects because of the risk to take wrong decisions based on their own and the developer’s perceptions.

Keywords. Requirements, Volatility Measures, Empirical Validation, Case Study, Use Case Model.
5.1 Introduction

Requirements development is a learning, rather than a gathering, process. As a consequence, requirements change frequently, even during later stages of the development process. These changes have several impacts on the software development life cycle [44].

The concept of requirements volatility is not well defined. In the Oxford Dictionary, the term "volatile" is defined as “easily changing” [20]. We define requirements volatility as the amount of changes to a use case model over time, a definition similar to those proposed in [31, 37, 39].

Volatile requirements can cause cost and schedule overruns making the goals of the project hard to achieve. Studies show that requirements volatility has a high impact on project performance [40, 44]. Several aspects of requirements volatility have been studied, for example its impact on project or process performance [2, 14, 21, 30, 33, 39, 40, 44], assessment and prediction [1, 7, 8, 19], simulation models [14, 33], and sources and causes of volatility [17, 31, 39, 40]. Zowghi and Numuliani [44] perform an empirical study on requirements volatility and its impact on project performance. They measure the perceived volatility by the developers in different phases of the software development. Their results show that frequent communication between users and developers has a positive impact on the stability of the requirements. The aspect we are most interested in is measures predicting requirements volatility. Software measurement can help us in providing guidance to the requirements management activities by quantifying and predicting changes to requirements. Predicting volatility can help project managers to take appropriate actions in order to minimise project risks and to set up a more stable process.

Table 1 summarizes some measures related to requirements volatility proposed in the literature. These are measures that are applicable, if the specific requirements are well-documented. However, even in the case of well-documented requirements, measures have to be tailored towards the particular organisation because each company has its own way of documenting requirements. Among the measures in table 1, only Ambriola and Gervasi [1] describe an empirical validation of the measures, showing that the stability measure had a high predictive value of risky trends in a requirements analysis process. The reason for validating measures is to empirically demonstrate their practical utility [13, 23, 38]. A measure is empirically valid, if there is a consistent relationship between the measure and an external attribute [45]. To our knowledge, no empirical validation on requirements volatility measures has been performed in an industrial setting.

Industrial studies investigate real projects, with professional developers and real customers. In such studies, it is often difficult to get information because of confidentiality and because research is not a high priority. Therefore, it is hard to control variables.
In this paper, we describe an industrial case study that investigated the requirements volatility of a diagnostic software system. The measures defined in the study were associated to the internal attributes size of use case model (UCM) and size of change to UCM (see table 2). The measures are a subset of those defined in [27], obtained by applying the goal question metrics (GQM) [3, 41] to the requirements management key process area of the capability maturity model [32]. The actual subset used for this case study has been tailored towards the specific company.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ambriola and Gervasi [1]</td>
<td>Amount of information contained in requirements at a certain time.</td>
</tr>
<tr>
<td>Costello and Liu [10]</td>
<td>Number of changes (addition, deletions, modifications) classified by reasons for changes in a given time interval; cumulative number of changes; total number of requirements.</td>
</tr>
<tr>
<td>Harker and Eason [17]</td>
<td>Stable requirements; changing requirements classified in mutable, emergent, consequential, adaptive, migration.</td>
</tr>
<tr>
<td>Henderson -Sellers et al. [18]</td>
<td>Use case size measures; environmental factors; total number of atomic actions in all flows and alternative flow; number of atomic actions per goal and actor; number of goals per stakeholder.</td>
</tr>
<tr>
<td>Henry and Henry [19]</td>
<td>Number of specification change; for each specification change: average changed SLOC, average changed modules, average change SLOC per module, average SLOCs/person/day.</td>
</tr>
<tr>
<td>Hammer et al. [16]</td>
<td>Total number of new requirements; modification to requirements; requirements traceability.</td>
</tr>
<tr>
<td>Javed et al. [21]</td>
<td>Pre/post functional specification changes; and post release changes.</td>
</tr>
<tr>
<td>Lam and Shankararaman [25]</td>
<td>Change effort; change volatility; change completeness; change error rate; requirements change density.</td>
</tr>
<tr>
<td>Malaiya and Denton [30]</td>
<td>Requirements changes in time; additions, deletions, and modifications to software.</td>
</tr>
<tr>
<td>Nurmuliani et al. [31]</td>
<td>Change types (addition deletion, modification); reason of change; origin.</td>
</tr>
<tr>
<td>Raynus [34]</td>
<td>Total number of system requirements; number of requirements added, modified, deleted; percentage of total requirements changes.</td>
</tr>
<tr>
<td>Rosenberg and Hyatt [37]</td>
<td>The percentage of requirements changed in a given time period</td>
</tr>
<tr>
<td>Stark et al. [39, 40]</td>
<td>Type of requirements; the planned and actual effort days for each requirement; the planned and actual number of calendar days for a version; requirements changes made to the version after plan approval (i.e. type of change, requesting group, and impact).</td>
</tr>
</tbody>
</table>

**TABLE 6:** Measures used in requirements volatility literature
The study was performed at Land Systems Hägglunds (BAE Systems), Sweden, with two goals: 1) to empirically validate a set of measures associated with the external process attribute requirements volatility; 2) to investigate the correlations between the subjective volatility and the objective volatility measured through size of change to UCMs (see table 2). We collected data in retrospect on fourteen UCMs (comprising 39 use cases) for a medium size software project and interviewed the stakeholders of the project (professional developers) about requirements volatility.

In the remainder of this paper, we present the case study in section 2 and the conclusions in section 3.

5.2 Case study description

We followed the guidelines of Wohlin et al. [42] and Kitchenham et al. [22]. All materials of the study are available on-line to enable replication [28].

Following the GQM template [3, 41] for the definition of the goals, we obtain the following definition for our case study: Analyse the use case models of a diagnostic software system. For the purpose of 1) empirically validating requirements measures and 2) investigating the correlation between the subjective and objective measures. With respect to volatility of requirements. From the point of view of two researchers at Umeå university. In the context of the company Land System Hägglunds.

### TABLE 7: Internal attributes and measures of volatility

<table>
<thead>
<tr>
<th>Size of UCM</th>
<th>Size of change to UCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of lines</td>
<td>total number of changes</td>
</tr>
<tr>
<td>number of words</td>
<td>number of minor changes</td>
</tr>
<tr>
<td>number of actors</td>
<td>number of moderate changes</td>
</tr>
<tr>
<td>number of use cases</td>
<td>number of major changes</td>
</tr>
<tr>
<td></td>
<td>number of revisions</td>
</tr>
</tbody>
</table>

The first goal of the study is to validate a set of requirements volatility measures empirically. A measure is valid, if it is internally valid [13] and part of a prediction system of the kind $Y= f(X)$, where $Y$ is the external attribute, i.e. the dependent variable and $X$ is the internal attribute i.e. the independent variable. According to Zuse [45], we have to find correlations and prove the causality between the dependent variable and the independent variables. In our case study the dependent variable is volatility measured through size of change to UCM, and the independent variable is size of UCM (see table 2 for the measures). However, we can only test for correlation (and not causality) due to low control compared to formal experiments.

The second goal serves to evaluate the precision of estimations of requirements volatility by project members and managers (in past projects). This is important because project managers and developers often make predictions on future projects and eventually take decisions based on their estimations.
Planning

Context selection

The case study was performed in retrospect at Land Systems Hägglunds (BAE Systems) Sweden. The company produces automotive systems with embedded software and has ISO9001 and ISO14001 certifications. The project chosen for the study (host project), builds external diagnostics software that runs on personal computers. At first, this software was used in another project at the company. Later on, the software was used also by external customers. Ten people have worked on the host project: two project managers, and eight developers. The software development process used was Rational Unified Process (RUP) [24]. Two iterations were performed in the elaboration phase, four in the construction phase, and one each in the inception and transition phases. The system comprises fourteen UCMs created in the inception phase (see table 3). According to the project terminology, a file containing a UCM is made up of an introduction, a revision table, a use case diagram, and a description of all actors and use cases (see [28] for a template of a UCM). As can be observed in figure 1, use case modelling was the only technique used in the host project to describe requirements (besides the vision document, which contains only sketchy requirements). The abstraction level of the

UCMs was high enough to be considered as requirements and not initial designs. Drafts of the UCMs were used to communicate requirements to the external customer. There were also

FIGURE 16. Requirements and design phases at the company
eight non-functional requirements. Since we did not have historical data available about
them, we did not consider them for the study. The project started in March 2001 and was
completed in August 2003. The time delay between the project and the case study was about
one year. The context of the case study can briefly be characterized according to four different
dimensions: 1) online, because it is performed in an industrial software development environ-
ment; 2) professional, i.e. non-student environment; 3) real, because it addresses a real prob-
lem; 4) specific, since it is focused on particular (volatility) measures.

In order to determine the subjective volatility, we contacted ten stakeholders at the com-
pany. Among them were two project managers (of which one was also a developer), three de-
velopers and five internal users of the software system. The project manager in charge of the
project helped us in identifying suitable subjects for our study. In the host project, the internal
users were part of a "reference group" consisting of several stakeholders. The task of this group
was to review the requirements specifications, and to act as the internal customer representa-
tive. All questions with impact on the requirements were managed by this group. External
customers were not interviewed, since they were only familiar with initial versions of the UCMs.
According to the project manager, they have never seen the complete use case models,
only short descriptions of them.

The objects of the study were fourteen UCMs of the project at the company, comprising
39 use cases in total. Other documentation was used to gather data about the objects, in par-
ticular project plans, iteration plans, and test plans. The guidelines for the subjects were de-
scribed in an email sent to all participants (see [28]). The measurement instruments used were
Microsoft Excel forms and Minitab (a software package for statistical analysis).

<table>
<thead>
<tr>
<th>UC model</th>
<th>Number of revisions</th>
<th>Total number of changes</th>
<th>Number of minor changes</th>
<th>Number of moderate changes</th>
<th>Number of major changes</th>
<th>Number of actors</th>
<th>Number of use cases</th>
<th>Number of lines</th>
<th>Number of words</th>
<th>Subjective volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ucm1</td>
<td>4</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>99</td>
<td>750</td>
<td>0.354</td>
</tr>
<tr>
<td>Ucm2</td>
<td>7</td>
<td>12</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>111</td>
<td>657</td>
<td>0.708</td>
</tr>
<tr>
<td>Ucm3</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>64</td>
<td>218</td>
<td>0.25</td>
</tr>
<tr>
<td>Ucm4</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>98</td>
<td>792</td>
<td>0.396</td>
</tr>
<tr>
<td>Ucm5</td>
<td>9</td>
<td>11</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>102</td>
<td>498</td>
<td>0.5</td>
</tr>
<tr>
<td>Ucm6</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>94</td>
<td>670</td>
<td>0.604</td>
</tr>
<tr>
<td>Ucm7</td>
<td>8</td>
<td>13</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>139</td>
<td>937</td>
<td>0.458</td>
</tr>
<tr>
<td>Ucm8</td>
<td>5</td>
<td>19</td>
<td>2</td>
<td>17</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>119</td>
<td>763</td>
<td>0.479</td>
</tr>
<tr>
<td>Ucm9</td>
<td>7</td>
<td>10</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>73</td>
<td>350</td>
<td>0.208</td>
</tr>
<tr>
<td>Ucm10</td>
<td>8</td>
<td>21</td>
<td>2</td>
<td>16</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>143</td>
<td>928</td>
<td>0.458</td>
</tr>
<tr>
<td>Ucm11</td>
<td>8</td>
<td>14</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>106</td>
<td>641</td>
<td>0.521</td>
</tr>
<tr>
<td>Ucm12</td>
<td>8</td>
<td>19</td>
<td>3</td>
<td>15</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>163</td>
<td>1068</td>
<td>0.271</td>
</tr>
<tr>
<td>Ucm13</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>58</td>
<td>216</td>
<td>0.25</td>
</tr>
<tr>
<td>Ucm14</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>67</td>
<td>283</td>
<td>0.292</td>
</tr>
<tr>
<td>Totals</td>
<td>90</td>
<td>161</td>
<td>32</td>
<td>116</td>
<td>13</td>
<td>5 (unique actors only)</td>
<td>39</td>
<td>1436</td>
<td>8771</td>
<td>not applicable</td>
</tr>
</tbody>
</table>
Attributes and measures

The state and response variables for the first goal were size of UCM and size of change to UCM, respectively. The measures of the two internal attributes (described in table 2) are quite intuitive, except for number of revisions. A revision of a UCM is a version of a UCM with a unique identifier. Revisions to UCMs are done when changes are performed, but also to validate the UCMs. A revision can include several changes or no changes at all. Usually, a large amount of revisions corresponds to a large amount of changes. However, almost all UCMs have one revision (usually the last one) with no changes associated. Among the 90 revisions, we counted 11 revisions with no changes (besides the initial one for each UCM which also have no changes associated). One possible measure of volatility could also be “number of associations between use cases”, connected to the internal attributes complexity and size of use case model. We discarded this measure because most UCMs in the host project did not have any associations.

In the second goal, the state variable was the internal attribute size of change to UCM, while the response variable was the external attribute subjective volatility. Subjective volatility was measured by subject ratings, which were collected manually by an e-mailed form.

Although we are aware of the difficulty of determining the exact scale type [6], our measures seem to belong to the ratio scale. The rules defined for measuring the UCMs are described in [28].

Hypotheses formulation

Our hypotheses were the following:

- There is significant correlation between the size of change to UCM and the size of UCM.
- There is significant correlation between the measures of size of change to UCM and size of UCM and the rating of volatility of the UCMs made by the subjects.

To test the first hypothesis we chose correlation [4]. For the second hypothesis we chose a within-subject design (i.e. all subjects filled in the same form).

Executing the study

The preparation of the subjects was made by explaining the definition of requirements volatility, describing the forms, and showing an example of how the form should be filled in [28]. The subjects were not aware of the hypotheses of the study. We handed in the material with the fourteen UCMs of the project, providing only their descriptive names. Each subject worked alone and could use unlimited time. They rated the relative volatility defined as trends in changes to UCMs in the phases inception, elaboration, construction, and transition on a scale of three linguistic labels (low, medium, and high). Based on [11], we chose three linguistic labels because an odd number of possible outcome yields better results due to the fact that there is a single medial value. This allowed us to create a form that fitted on to a single page and was easy and quick to fill in, in order to encourage the subjects to participate in the study. We asked the subjects not to read the documentation of the UCMs in order to
avoid letting the objective volatility affect the subjective volatility. This documentation contains some information (like the number of revisions, the description and the date of changes to the UCM) which may reveal the objective volatility of the UCMs.

The study was divided into two phases: the first part consisted of manual collection of data for the measures in table 2. We gathered data starting from the first available revision of the UCMs, which were dated July 2001 (three months after the beginning of the project). Data was collected by studying historical project documentation. In the second phase we distributed the forms. We contacted ten stakeholders at the company for the subjective data, and received answers from eight of them. Two of the forms were discarded because the data was considered unreliable (the same answer was given for all questions). The study did not affect the development project because it was done in retrospect. The data is described in table 3.

Analysis and interpretation

To verify the two hypotheses we first checked the distributions of data for normality. As the data distribution were not normal, we used non-parametric statistics and applied the Spearman correlation coefficient. We chose a level of significance $\alpha = 0.05$, i.e. the level of confidence is 95%. For a sample of size fourteen, the Spearman cutoff for accepting $H_0$ is 0.532 [43]. However, as the formality in our case study is low compared to formal experiments, we consider the Spearman cutoff only a reference point to judge the significance of our correlations.

Analysis of the first hypothesis

All size of change to UCM measures were correlated separately with the size of UCM measures. As we can observe in table 4, the values in bold show high correlation. The measure total number of changes is highly correlated with the measures of size of UCM. The number of minor changes is not correlated at all, but this is not surprising, because we have defined minor changes as changes in the style or structure of the file. These changes do not affect the size of UCM. The other measures seem to be somewhat correlated. Concluding, there is a high correlation between the size of UCM and the total number of changes to it. For the moment,
we do not claim any causality (i.e. that larger use cases cause a higher number of changes). Further studies are needed to check the causality and to identify guidelines for optimal size of UCMs from a volatility perspective.

**Analysis of the second hypothesis**

Since the rating of volatility made by the subjects yielded ordinal data we applied a simple transformation (weighted average) to obtain ranked data. For the current analysis, we averaged the answers for the phases to one value for each UCM. Each of the measures was correlated separately with the transformed rating of volatility made by the subjects (last row in table 4). Because all coefficients are below the cutoff, we can conclude that there is no significant correlation between our measures and the subjective rating of volatility. It is interesting to note that measures like total number of changes, which is largely utilised to measure volatility [10, 21, 30], did not show very high correlation. This may be due to the small size of the sample. Another reason can be that the subjective volatility by the stakeholders could be affected by other parameters. The stakeholders could perceive as highly volatile those UCMs affected by frequent changes very late in the development process, since the late changes are closest to the time of the case study. This could be tested by checking the transition phase separately. The subjects might recall that there was some problem with a specific UCM but in reality the problems were in other phases of the software development and did not affect the functionality agreed upon with the customer, i.e. without affecting the specific UCM.

**Validity evaluation**

**Threats to conclusion validity**

One issue that could affect conclusion validity is the size of the sample data (fourteen UCMs and six subjects). If the sample size is small, non-parametric tests can lead to accept the null hypothesis [6]. Therefore, we consider our results as preliminary.

The objective data have been calculated using a computerized tool and are therefore reliable, except the categorization of changes as minor, moderate, and major respectively, because human judgement was involved. However, we have defined measurement rules to keep the judgement as objective as possible [28].

**TABLE 9: Spearman correlation coefficient obtained analysing the two hypotheses**

<table>
<thead>
<tr>
<th></th>
<th>Total number of changes</th>
<th>Number of minor changes</th>
<th>Number of moderate changes</th>
<th>Number of major changes</th>
<th>Number of revisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of lines</td>
<td>0.909</td>
<td>0.322</td>
<td>0.675</td>
<td>0.579</td>
<td>0.573</td>
</tr>
<tr>
<td>Number of words</td>
<td>0.675</td>
<td>0.410</td>
<td>0.523</td>
<td>0.473</td>
<td>0.240</td>
</tr>
<tr>
<td>Number of actors</td>
<td>0.632</td>
<td>0.270</td>
<td>0.497</td>
<td>0.129</td>
<td>0.427</td>
</tr>
<tr>
<td>Number of use cases</td>
<td>0.757</td>
<td>0.457</td>
<td>0.412</td>
<td>0.731</td>
<td>0.600</td>
</tr>
<tr>
<td>Subjective volatility</td>
<td>0.433</td>
<td>0.255</td>
<td>0.383</td>
<td>0.123</td>
<td>0.267</td>
</tr>
</tbody>
</table>
The participants of the study formed an heterogeneous group. We could have chosen a homogenous group with the disadvantages of decreasing the number of subjects and affecting external validity. However, if the group is very heterogeneous the variations due to the differences among the subjects is bigger than the variations due to the treatments, which actually is a threat in our case.

**Threats to construct validity**

The subjective volatility is based on judgement of the subjects. The subjects chosen were all involved in the host project as developers or internal users. These users also had the task of reviewing requirements. Therefore, we believe that they were capable of accurately and reliably completing the form. To measure more precisely the reliability of the subjects, we evaluated the intrarater agreement by applying the Cohen's Kappa method [9]. The Kappa value ranges from 0 (no agreement) to 1 (full agreement). We obtained 0.54 which is considered a moderate strength of agreement on the four benchmarks mentioned in [12]. Details of how we obtained this value are available on-line (see [28]).

We consider construct validity of the measures in table 2 somewhat ensured, since we used the GQM [3, 41] to define the measures and we theoretically validated them [29].

**Threats to internal validity**

Differences among subjects. Because the group of subjects was heterogeneous, error variance due to differences among subjects is reduced. Our subjects had different background, but it was not necessary to have previous knowledge about requirements engineering to be able to fill in the forms distributed. Therefore this threat was considered small.

Knowledge of the domain. All UCMs belonged to the same universe of discourse, and the knowledge of the domain of the project did not affect internal validity.

Accuracy of collected data. The changes to requirements were not well documented. They were determined by manually comparing several versions of files. Rules of measurements were defined. However, there is a risk that the way we defined changes to requirements can be different from the subjects’ view of change.

Accuracy of responses. One factor affecting the reliability of the answers can be the time that had passed since the end of the project. However, the best time to collect reliable data and reduce the recall bias is debatable. Considering that the average length of a typical software development project at the company is about three years, one year of delay in a project context is not a long time. Furthermore, the subjects work on very few projects in parallel. Other factors affecting the subjective measures can be personal problems and mood. The developers might not have read the definition of volatility carefully and answered the form randomly. The developers might even have read the documentation related to the UCMs, affecting the perceived volatility by studying the objective data. However, this threat is small because it takes time to read the documentation of all the UCMs and we believe that the developers were not willing to spend that time. This would have been a threat in case of correlation.
Motivation of subjects. Due to the small size of the form we believed that it was not necessary to motivate the subjects. Only 75% of the answers were valid, but the subjects most heavily involved in the project did answer.

Other factors. To fill in the form required less than 30 minutes, therefore fatigue effects were not a relevant factor. Plagiarism could be checked easily, while influence among the subjects could not be controlled for and we could only trust the answers received.

Threats to external validity
Our threats to external validity are minimal because the study was performed in an industrial environment. Therefore, the materials used and the project were real, and the subjects were professionals. The only threat could be the relatively small size of the project.

5.3 Conclusions
In this paper, we have described a retrospective case study on requirements volatility performed at Land Systems Hägglunds (BAE Systems), Sweden. We collected different measures of size of UCM and size of change to UCMs, associated to the external process attribute requirements volatility for a medium size software project. We furthermore interviewed project stakeholders about their perception of requirements volatility.

Analysing the results for the first hypothesis (empirical validation), we found that the measure total number of changes to UCMs is highly correlated with the size of UCM. The measures of size of UCM are validated in the specific environment and can be considered good indicators of volatility. This result supports the intuitively viable notion that larger UCMs are more volatile than smaller ones and should encourage developers to modularize UCMs. This serves to re-emphasise some fundamental software engineering principles for the need of modularity and cohesion in order to manage complexity and localize change. However, it is not clear yet, whether there is a linear relationship between the size and the number of changes to UCMs. Further studies are needed to identify guidelines for optimal size of UCMs from a volatility perspective. The results are limited to one project and therefore cannot be generalised.

Analysing the results for the second hypothesis, we could not find significant correlations between any of our measures and the rating of volatility by the subjects. The important result is that perception did not match our measures. This is somewhat surprising, since measures of changes to requirements are suggested as reliable indicators of requirements volatility in the literature. There are many possible explanations for our results and further investigations are needed. It might for example be possible that the subjects did not reliably recall the evolution of all UCMs. Further factors might have contributed to subjective volatility, like the actual impacts of changes, priorities of use cases, or functional details in general. Even things that have nothing to do with requirements per se as for example changing design decisions. This implies that decision makers may take a high risk when basing decisions solely on subjective requirements volatility. Therefore, we suggest that project managers at this company measure their projects in order to minimize this risk.
5.4 References


### 5.5 Appendices

Objective data collected on fourteen use case models

<table>
<thead>
<tr>
<th>Revision ID</th>
<th>Revision Date</th>
<th>What is Changed</th>
<th># Changes</th>
<th>Date on Head.</th>
<th>Factors</th>
<th>Suc</th>
<th>Smiles</th>
<th>Size of Change</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td>0</td>
<td>0</td>
<td>2001-07-26</td>
<td>1</td>
<td>3</td>
<td>83</td>
<td>0</td>
<td>680</td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-24</td>
<td>2rew.text, RMuc</td>
<td>3</td>
<td>2001-08-24</td>
<td>1</td>
<td>4</td>
<td>78</td>
<td>0</td>
<td>599</td>
</tr>
<tr>
<td>PA3</td>
<td>2001-09-17</td>
<td>header, 2rew.text, RMucDel</td>
<td>4</td>
<td>2001-09-17</td>
<td>1</td>
<td>3</td>
<td>96</td>
<td>0</td>
<td>741</td>
</tr>
<tr>
<td>A</td>
<td>2001-10-01</td>
<td>header</td>
<td>1</td>
<td>2003-02-12</td>
<td>1</td>
<td>3</td>
<td>99</td>
<td>0</td>
<td>730</td>
</tr>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td>0</td>
<td>0</td>
<td>2001-07-26</td>
<td>1</td>
<td>2</td>
<td>60</td>
<td>0</td>
<td>344</td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-24</td>
<td>vision, RMactor, RMuc</td>
<td>5</td>
<td>2001-08-27</td>
<td>1</td>
<td>2</td>
<td>68</td>
<td>0</td>
<td>599</td>
</tr>
<tr>
<td>PA3</td>
<td>2001-09-17</td>
<td>Header, RMgeneral,RMucactor</td>
<td>3</td>
<td>2001-09-17</td>
<td>1</td>
<td>1</td>
<td>51</td>
<td>0</td>
<td>142</td>
</tr>
<tr>
<td>A</td>
<td>2001-10-02</td>
<td>RM general, RM uc</td>
<td>2</td>
<td>2001-10-02</td>
<td>1</td>
<td>2</td>
<td>88</td>
<td>0</td>
<td>547</td>
</tr>
<tr>
<td>PB1</td>
<td>2002-08-06</td>
<td>header, RMucAdded</td>
<td>2</td>
<td>2002-08-09</td>
<td>1</td>
<td>3</td>
<td>103</td>
<td>0</td>
<td>635</td>
</tr>
<tr>
<td>PB2</td>
<td>2002-09-19</td>
<td>RMuc</td>
<td>1</td>
<td>2002-09-19</td>
<td>1</td>
<td>3</td>
<td>107</td>
<td>0</td>
<td>651</td>
</tr>
<tr>
<td>B</td>
<td>2003-02-12</td>
<td>0</td>
<td>0</td>
<td>2003-02-12</td>
<td>1</td>
<td>3</td>
<td>111</td>
<td>0</td>
<td>657</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revision ID</th>
<th>Revision Date</th>
<th>What is Changed</th>
<th># Changes</th>
<th>Date on Head.</th>
<th>Factors</th>
<th>Suc</th>
<th>Smiles</th>
<th>Size of Change</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td>0</td>
<td>0</td>
<td>2001-07-24</td>
<td>1</td>
<td>2</td>
<td>45</td>
<td>0</td>
<td>186</td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-24</td>
<td>style, RMactor</td>
<td>2</td>
<td>2001-08-27</td>
<td>1</td>
<td>2</td>
<td>48</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>PA3</td>
<td>2001-09-17</td>
<td>header, RMUC</td>
<td>3</td>
<td>2001-09-17</td>
<td>1</td>
<td>2</td>
<td>94</td>
<td>0</td>
<td>262</td>
</tr>
<tr>
<td>A</td>
<td>2001-10-01</td>
<td>header, RMfigure</td>
<td>2</td>
<td>2003-02-12</td>
<td>1</td>
<td>2</td>
<td>98</td>
<td>0</td>
<td>592</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revision ID</th>
<th>Revision Date</th>
<th>What is Changed</th>
<th># Changes</th>
<th>Date on Head.</th>
<th>Factors</th>
<th>Suc</th>
<th>Smiles</th>
<th>Size of Change</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td>0</td>
<td>0</td>
<td>2001-07-24</td>
<td>1</td>
<td>2</td>
<td>45</td>
<td>0</td>
<td>186</td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-24</td>
<td>style, RMactor</td>
<td>2</td>
<td>2001-08-27</td>
<td>1</td>
<td>2</td>
<td>48</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>PA3</td>
<td>2001-09-17</td>
<td>header, RMUC</td>
<td>3</td>
<td>2001-09-17</td>
<td>1</td>
<td>2</td>
<td>94</td>
<td>0</td>
<td>262</td>
</tr>
<tr>
<td>A</td>
<td>2001-10-01</td>
<td>header, RMfigure</td>
<td>2</td>
<td>2003-02-12</td>
<td>1</td>
<td>2</td>
<td>98</td>
<td>0</td>
<td>592</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Revision ID</th>
<th>Revision Date</th>
<th>What is Changed</th>
<th># Changes</th>
<th>Date on Head.</th>
<th>Factors</th>
<th>Suc</th>
<th>Smiles</th>
<th>Size of Change</th>
<th>Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td>0</td>
<td>0</td>
<td>2001-07-26</td>
<td>1</td>
<td>2</td>
<td>62</td>
<td>0</td>
<td>456</td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-27</td>
<td>vision, Rosemodelgeneral, actor</td>
<td>4</td>
<td>2001-08-27</td>
<td>1</td>
<td>2</td>
<td>73</td>
<td>0</td>
<td>599</td>
</tr>
<tr>
<td>PA3</td>
<td>2001-09-17</td>
<td>header, UC</td>
<td>2</td>
<td>2001-09-17</td>
<td>1</td>
<td>2</td>
<td>79</td>
<td>0</td>
<td>602</td>
</tr>
<tr>
<td>A</td>
<td>2001-10-01</td>
<td>rose model rew text, UC rew text</td>
<td>2</td>
<td>2001-10-01</td>
<td>1</td>
<td>2</td>
<td>85</td>
<td>0</td>
<td>642</td>
</tr>
<tr>
<td>PB1</td>
<td>2002-01-22</td>
<td>header, UC</td>
<td>2</td>
<td>2002-01-22</td>
<td>1</td>
<td>2</td>
<td>89</td>
<td>0</td>
<td>668</td>
</tr>
<tr>
<td>B</td>
<td>2003-02-12</td>
<td>0</td>
<td>0</td>
<td>2003-02-12</td>
<td>1</td>
<td>2</td>
<td>94</td>
<td>0</td>
<td>670</td>
</tr>
<tr>
<td>Revision ID</td>
<td>Revision Date</td>
<td>What is Changed</td>
<td># Changes</td>
<td>Date on Header</td>
<td># Actors</td>
<td># Lines</td>
<td>Size of Change</td>
<td># Words</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>----------------</td>
<td>-----------</td>
<td>----------------</td>
<td>----------</td>
<td>---------</td>
<td>----------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td>vision, 2actor, UC</td>
<td>4</td>
<td>2001-07-24</td>
<td>2</td>
<td>3</td>
<td>89</td>
<td>598</td>
<td></td>
</tr>
<tr>
<td>PA1</td>
<td>2001-07-17</td>
<td>header, UC</td>
<td>2</td>
<td>2001-09-17</td>
<td>2</td>
<td>4</td>
<td>104</td>
<td>727</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2001-10-02</td>
<td>vision, 2actor, UC</td>
<td>0</td>
<td>2001-10-02</td>
<td>2</td>
<td>4</td>
<td>108</td>
<td>736</td>
<td></td>
</tr>
<tr>
<td>PB1</td>
<td>2001-12-04</td>
<td>UC, UC</td>
<td>2</td>
<td>2001-12-04</td>
<td>2</td>
<td>4</td>
<td>114</td>
<td>754</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2001-12-17</td>
<td>UC, header</td>
<td>1</td>
<td>2001-12-17</td>
<td>1</td>
<td>2</td>
<td>117</td>
<td>768</td>
<td></td>
</tr>
<tr>
<td>PC1</td>
<td>2002-05-27</td>
<td>Rosen model figures, UC</td>
<td>2</td>
<td>2002-05-27</td>
<td>2</td>
<td>4</td>
<td>134</td>
<td>931</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2003-02-12</td>
<td></td>
<td>2</td>
<td>2003-02-12</td>
<td>2</td>
<td>4</td>
<td>139</td>
<td>937</td>
<td></td>
</tr>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td>vision, 2actor, UC</td>
<td>4</td>
<td>2001-07-24</td>
<td>2</td>
<td>3</td>
<td>89</td>
<td>598</td>
<td></td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-28</td>
<td>vision, 2actor, UC</td>
<td>6</td>
<td>2001-08-28</td>
<td>2</td>
<td>3</td>
<td>84</td>
<td>655</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2001-10-01</td>
<td>RM figure</td>
<td>1</td>
<td>2001-10-01</td>
<td>1</td>
<td>1</td>
<td>58</td>
<td>283</td>
<td></td>
</tr>
<tr>
<td>PB1</td>
<td>2003-05-13</td>
<td>UC</td>
<td>1</td>
<td>2003-05-12</td>
<td>1</td>
<td>3</td>
<td>119</td>
<td>763</td>
<td></td>
</tr>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td></td>
<td>0</td>
<td>2001-07-24</td>
<td>2</td>
<td>3</td>
<td>74</td>
<td>531</td>
<td></td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-27</td>
<td>vision, 2actor, UC</td>
<td>6</td>
<td>2001-08-27</td>
<td>2</td>
<td>3</td>
<td>86</td>
<td>587</td>
<td></td>
</tr>
<tr>
<td>PB1</td>
<td>2002-05-13</td>
<td>UC</td>
<td>1</td>
<td>2002-05-13</td>
<td>1</td>
<td>1</td>
<td>61</td>
<td>317</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2001-10-01</td>
<td>vision, 2RM general</td>
<td>3</td>
<td>2002-09-09</td>
<td>1</td>
<td>1</td>
<td>69</td>
<td>344</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2003-02-12</td>
<td></td>
<td>0</td>
<td>2003-02-12</td>
<td>2</td>
<td>3</td>
<td>73</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td></td>
<td>0</td>
<td>2001-07-26</td>
<td>2</td>
<td>3</td>
<td>74</td>
<td>531</td>
<td></td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-27</td>
<td>RM rew text, 2actor, rew text</td>
<td>3</td>
<td>2001-08-27</td>
<td>2</td>
<td>3</td>
<td>78</td>
<td>488</td>
<td></td>
</tr>
<tr>
<td>PA3</td>
<td>2001-09-17</td>
<td>header, 2RM general, UC</td>
<td>4</td>
<td>2001-09-17</td>
<td>2</td>
<td>4</td>
<td>86</td>
<td>589</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2001-10-01</td>
<td>RM figure</td>
<td>1</td>
<td>2001-10-01</td>
<td>1</td>
<td>1</td>
<td>58</td>
<td>283</td>
<td></td>
</tr>
<tr>
<td>PB1</td>
<td>2002-03-11</td>
<td>header, RM gen, 5UC</td>
<td>7</td>
<td>2002-03-11</td>
<td>2</td>
<td>6</td>
<td>122</td>
<td>863</td>
<td></td>
</tr>
<tr>
<td>PB2</td>
<td>2002-05-13</td>
<td>UC</td>
<td>1</td>
<td>2002-05-13</td>
<td>2</td>
<td>6</td>
<td>128</td>
<td>924</td>
<td></td>
</tr>
<tr>
<td>PB3</td>
<td>2002-08-09</td>
<td>2UC</td>
<td>2</td>
<td>2002-08-09</td>
<td>2</td>
<td>6</td>
<td>138</td>
<td>922</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2003-02-12</td>
<td></td>
<td>0</td>
<td>2003-02-12</td>
<td>2</td>
<td>6</td>
<td>143</td>
<td>929</td>
<td></td>
</tr>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td></td>
<td>0</td>
<td>2001-07-24</td>
<td>2</td>
<td>3</td>
<td>72</td>
<td>445</td>
<td></td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-27</td>
<td>RM rew text, 2actor, rew text</td>
<td>3</td>
<td>2001-08-27</td>
<td>2</td>
<td>3</td>
<td>78</td>
<td>488</td>
<td></td>
</tr>
<tr>
<td>PA3</td>
<td>2001-09-17</td>
<td>header, RM gen, UC</td>
<td>5</td>
<td>2001-09-17</td>
<td>2</td>
<td>3</td>
<td>82</td>
<td>527</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2001-10-01</td>
<td>rose model general, text added</td>
<td>1</td>
<td>2001-10-01</td>
<td>2</td>
<td>3</td>
<td>90</td>
<td>562</td>
<td></td>
</tr>
<tr>
<td>PB1</td>
<td>2002-03-22</td>
<td>header, 2UC rew text</td>
<td>3</td>
<td>2002-03-22</td>
<td>2</td>
<td>3</td>
<td>94</td>
<td>597</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2002-02-11</td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>PB1</td>
<td>2002-08-09</td>
<td>rose model general, text added</td>
<td>1</td>
<td>2002-08-09</td>
<td>2</td>
<td>3</td>
<td>102</td>
<td>635</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>2003-02-12</td>
<td></td>
<td>0</td>
<td>2003-02-12</td>
<td>2</td>
<td>3</td>
<td>106</td>
<td>641</td>
<td></td>
</tr>
<tr>
<td>PB1</td>
<td>2002-09-19</td>
<td>UC</td>
<td>1</td>
<td>2002-09-19</td>
<td>3</td>
<td>3</td>
<td>162</td>
<td>1062</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2003-02-12</td>
<td></td>
<td>0</td>
<td>2003-02-12</td>
<td>3</td>
<td>3</td>
<td>163</td>
<td>1068</td>
<td></td>
</tr>
</tbody>
</table>
Subjective Data

<table>
<thead>
<tr>
<th>UC models</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCM 1</td>
<td>low</td>
<td>medium</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>medium</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td></td>
<td>low</td>
<td>medium</td>
<td>high</td>
<td>high</td>
</tr>
</tbody>
</table>

Transition: elaboration construction inception
**Measurement rules**

In this section, we describe the rules we adopted for measuring the UCMs. For this study we considered UCMs as units of requirements. Each UCM was described in a single file. We had full access to the repository of the project and retrieved all existing revisions of all UCMs from the start of the project until the current date. We measured size of UCM in different ways: number of lines and number of words in the file, number of use cases and number of actors. We used the MS Word count tool to count lines and words. The measure number of changes was counted in several ways: by counting the number of changes by size (major, moderate, minor) and the number of revisions.

The size of change is usually dependent on the effort spent for the change or the number of artifacts impacted by the change. Unfortunately, this information was not available in the host project, therefore, size of change to UCM was estimated by the authors. The change was considered minor for changes in the header, style, or structure. Moderate when text was changed (rewording text), major when a use case or an actor was added or deleted.

Since we did not have a database of changes to UCMs, we counted the changes manually using the following counting procedures and rules. We compared two successive versions of a UCM file using the "track changes: compare documents" tool in MS Word. A sentence highlighted in red or several words changed in one paragraph were considered as one moderate change. Exceptions to this rule were the following.

- Changes in the table of contents, (because these changes are shown later in the same file);
- changes in the revision table and the date in the header; addition or deletion of an empty space.
- A change of position of an actor or use case in the file; substitutions of synonymous.
- When a figure appeared as deleted in the comparison, we checked whether there were actual differences. If not, we did not consider it as a change.

**Instruments**

**Instructions for the subjects**

**DEFINITIONS**

- Volatility = the amount of changes to a use case model over time.
- Relative volatility = the volatility of a use case model compared with other use case models.

For each of the UCMs, please indicate the relative volatility of the UCM during the whole life cycle of the project by placing a cross in the appropriate box. More concretely, what you need to do is to compare the amount of changes for each UCM with the amount of changes in average in a certain phase of the other UCMs and indicate by an X if you think that the UCM was more volatile compared with the others.

**EXAMPLE** (see table below): This means that in the inception phase, the amount of changes to UCM 1 is higher compared with UCM 2 and it is very high compared with UCM 3. In the elaboration phase, the amount of changes to UCMs 1 and 2 is higher compared with UCM 3. And so on.
Interrater agreement

The reliability of measures depends on many things, for instance the measurement instruments. The basic principle is that by measuring the same thing twice we obtain the same result. Naturally, measures where human judgement is involved are less reliable than others.

The perceived volatility was measured by an assessment of the UCMs by subjects. If the assessments are unreliable, there is a risk that we arrive at wrong conclusions. One way to measure the reliability of assessments is interrater agreement. Interrater agreement is concerned with the extent of agreement in the ratings given by independent assessors. The subjectivity in ratings will make it most unlikely that there is a perfect agreement. High interrater agreement is desirable to give credibility to the assessment results.

One of the most popular methods for evaluating the interrater agreement is Cohen’s Kappa used in software engineering by El Emam and very popular in the social and behavioural sciences.

The Kappa value ranges from 0 (no agreement) to 1 (full agreement). It can be used as a criterion for evaluating the quality of an assessment a posteriori.

Quantify agreement with kappa

We organised the scores in a contingency table. The variables being rated are the fourteen use case models. Since the variables being rated have 12 categories (3 for low, medium, high times four, the four phases of RUP), the contingency table will be a 12x12 table. The ratings of each of the fourteen UCMs will be entered in the table. The subjects are divided in two groups that rated the volatility of UCMs indipendently. Ideally the groups should be equally competent. We tried to make the groups homogeneus. Agreements between the two groups of raters will be placed in the diagonal.

Number of observed agreements: 33 (58.93% of the observations)
Number of agreements expected by chance: 5.8 (10.27% of the observations)
Kappa = 0.542
95% confidence interval: From 0.399 to 0.686
The strength of agreement is considered to be 'moderate'.

<table>
<thead>
<tr>
<th>UCM models</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCM 1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>UCM 2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>UCM 3</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

**TABLE 10:**
The calculations above only consider exact matches between observers. If the categories (A, B, C...) are ordered, you may also wish to consider close matches. In other words, if one observer classifies a subject into group B and the other into group C, this is closer than if one classifies into A and the other into D. The calculation of weighted kappa, below, assumes the categories are ordered and accounts for how far apart the two raters are.

Weighted Kappa = 0.872
Assessed this way, the strength of agreement is considered to be 'very good'.

UCM template (pdf)
1 Introduction: <Rose_Package.Name>

1.1 Vision: <Word_Heading.Text>

1.2 References

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>Issued By</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA1</td>
<td>2001-07-04</td>
<td></td>
<td>First version</td>
</tr>
<tr>
<td>PA2</td>
<td>2001-08-24</td>
<td></td>
<td>Second revision</td>
</tr>
<tr>
<td>PA3</td>
<td>2001-09-17</td>
<td></td>
<td>Revised after meeting 1</td>
</tr>
<tr>
<td>PA4</td>
<td>2001-10-01</td>
<td></td>
<td>Revision A, no other changes</td>
</tr>
<tr>
<td>PA5</td>
<td>2002-01-22</td>
<td></td>
<td>Changes to test performance</td>
</tr>
</tbody>
</table>

2 Rose model: <Rose_Package.Name>

![Rose Package Documentation]

Figure 1: UseCaseDiagram.Name

Usecasemodelname.doc

2.1 Actor: <Classes.Name>

2.2 Use-case: <UseCase.Name>

![UseCase Documentation]
Paper 6

Construction and validation of prediction models for number of changes to requirements

Annabella Loconsole, Jürgen Börstler

Abstract

In this paper we present a correlational study in which we assess the ability of five size measures to predict the number of changes to requirements for a medium size software project. Although, the number of changes does not measure volatility directly, it is an important basic measure that can be used to easily compute other measures, like change density or change frequency. The study is explorative, i.e. we analyse the data collected for our measures to find out the best predictor of number of changes. To our knowledge, no empirical validation of requirements change measures as predictors has been performed in an industrial setting. Based on the data collected from two industrial projects for five measures of size of requirements (number of actors, use cases, words, lines, and revisions), we have built and evaluated prediction models for number of changes to requirements. These models can help project managers to estimate the volatility of requirements and minimize the risks caused by volatile requirements, like schedule and costs overruns. We performed a cross systems validation. For our best model we calculated a pred(0.25)=0.5, which is better than the accuracy of common effort prediction models like for example COCOMO. Although our models are
likely to have only local validity, the general method for constructing the prediction models could be applied in any software development company. In an earlier study, we showed that decisions solely based on developer perception are unreliable. Predictions models, like the one presented here can help to mitigate that risk.

Key words: Requirement, Prediction Model, Empirical Validation, Correlational Study

6.1 Introduction

Requirements engineering is an important phase of software development, where the needs of the stakeholders are collected, developed, and documented. Requirements development is a learning rather than a gathering process. As a consequence, requirements change frequently, even during later stages of the development process. Software requirements that change often are usually said to be volatile. Studies show that requirements volatility has a high impact on project performance (Pfahl and Lebsanft, 2000; Stark et al., 1999; Zowghi and Nurmuliani, 2002). However, we cannot expect the requirements to be stable, even when requirements engineering tasks (such as elicitation, analysis, specification, and validation) are performed well. It is therefore important to carefully monitor and control the requirements throughout the software life cycle. Monitoring requirements volatility usually involves measuring trends or percentages of changes to requirements (see section 4.2). Anticipating a certain level of volatility project managers can take appropriate actions in order to decrease project risks.

In this paper, we describe a correlational study with the goal of empirically validating five measures of requirements size as predictors for the number of requirements changes. We built seven prediction models using data collected for a medium-size software project developed at BAE Systems Hågglunds AB, Sweden. We then evaluated the accuracy of five models by applying them on a set of data collected for a second project at the same company.

The results show that the best predictors of number of changes are the length 1 measures: number of lines and words. Other predictors of complexity and functionality were found less accurate.

The remaining part of the paper proceeds as follows: section 2 describes the research related to empirical validation of measures in general and work related to requirements volatility. In section 3 we briefly summarise our previous case study, which investigated the relationship between four measures of size of use case models and requirements volatility. Section 4 describes the goals, hypotheses, and data collected in the present empirical study. The data analysis and the resulting prediction models are described in section 5. Finally, discussions and conclusions are presented in section 6.

footnote: Size can be seen as composed of length, functionality, and complexity (Fenton and Pfleeger, 1996).
6.2 Related work

There is little empirical research in the area of requirements volatility. The majority of the published studies, evaluate the impact of requirements volatility on software projects (Stark et al., 1999; Zowghi and Nurmuliani, 2002), on software products (Henry and Henry, 1993), and defect density (Javed et al., 2004; Malaiya and Denton, 1999).

Many measures related to requirements volatility have been proposed in the literature. Measures to assess requirements stability 2 are presented by Ambriola and Gervasi (2000). They showed that requirements stability had a high predictive value for project risks in a requirements analysis process. Other measures (Costello and Liu, 1995; Henderson-Sellers et al., 2002; Huffman et al., 1998; Hyatt and Rosenberg, 1996; Malaiya and Denton, 1999; Nurmuliani et al., 2004; Raynus, 1999; Stark et al., 1999) concern requirements volatility. In most of these cases, volatility is defined as a property of the whole set of requirements of a project. We, on the other hand, are concerned with the changing nature of smaller units of requirements. That means there can be different levels of requirements volatility for different units of requirements within one and the same project. This gives project managers a more fine-grained tool for requirements management (see section 4.2 for further details).

Few correlational studies of requirements volatility have been published (Ambriola and Gervasi, 2000; Henry and Henry, 1993; Javed et al., 2004; Loconsole and B¨orstler, 2005; Stark et al., 1999). In all of them, except Loconsole and B¨orstler (2005), requirements volatility was chosen as independent variable, i.e. as a predictor of other attributes. Henry and Henry (1993) for example, propose measures to predict the impact of volatility on the software product. Similarly, Stark et al. (1999) present measures to predict the effects of changing requirements on costs and schedule. Javed et al. (2004) present measures of requirements change and their correlation with software defects.

footnote: Sometimes, the word stability is used instead of volatility. For instance, a definition of “degree of stability” of requirements is presented in (IEEE830-1998, 1998). Both terms are used conjunctly in (Huffman et al., 1998; Raynus, 1999). In our opinion, the words are antonyms.

The majority of the measures above are designed for well-specified and wellwritten requirements, using standardised documentation templates. However, even in the case of well-documented requirements, measures have to be tailored towards the particular organisation, because each company has its own way of documenting requirements. Among those measures, only Ambriola and Gervasi (2000) and Loconsole and B¨orstler (2005) provide an empirical validation. The reason for validating measures is to empirically demonstrate their practical utility, i.e. to show that there is a consistent relationship between the measure and an external attribute (Fenton and Pfleeger, 1996; Kitchenham et al., 1995; Schneidewind, 1992; Zuse, 1997). It is important to ensure that the data collected for a certain measure is related to the actual property investigated (the attribute to be measured). Otherwise time and money is spent for collecting useless data. To our knowledge, only one empirical validation
of requirements volatility measures has been performed in an industrial setting (Loconsole and Börstler, 2005). In that study we showed a high correlation between four measures of size and number of changes to use case models (see section 3 for further details).

The measures we are interested in are measures that can help to predict the number of changes to requirements and this, in turn, can be used to determine volatility. There are no studies on prediction models of requirements changes. Bush and Finkelstein (2002, 2003) describe a process that could support a predictive view of requirements stability. Starting from an initial set of requirements, this process helps to create worlds of possible evolutions of requirements. They also report on positive results from an industrial case study validating the approach. However, using their process is complex and time consuming. Our approach is much simpler and better suited for small and medium size companies. It is different from the other approaches, because usually volatility is analysed looking at the set of the requirements as a whole, at a higher abstraction level (project level). In our approach instead, we look at the volatility of the single requirement. It is based on use case requirements. Nevertheless, except use case diagrams, the requirements are mainly text based, therefore, it is possible to generalise our results. Further studies are needed to prove this.

6.3 Background
In an earlier industrial case study (Loconsole and Börstler, 2005), we investigated measures of volatility for a medium size software project. Our goals were: 1) to empirically validate a set of measures associated with the volatility of requirements documents; and 2) to investigate the correlation between perceived and measured volatility. We collected size and change data in retrospect for all versions of requirements documents of the software project. In addition, we determined the perceived volatility by interviewing stakeholders of the project.

The spearman correlation coefficient was calculated between each measure of size of requirements documents and the size of changes to requirements documents. Requirements in the project were described in terms of use cases and each requirements document contained one use case model. The size of a requirements document was measured in terms of number of lines, number of words, number of use cases, and number of actors.

The data analysis showed a high correlation between each of the size measures and the total number of changes. This suggests that our measures of size of requirements documents are good indicators of the number of changes for use case based requirements documents. For the second goal, we could not find significant correlations between any of our four volatility measures and the rating of volatility by the experts. This implies that the developers’ perceptions of number of changes were not good indicators of requirements volatility for the project analysed. These results suggest that managers at this company should measure their projects because of the risk to take wrong decisions based solely on their own and the developers perceptions.
6.4 Description of the correlational study

The goal of the present study is to analyse the ability of five specific measures to predict the number of changes to requirements, using two data sets. Prediction models were constructed applying univariate and multivariate linear regression to the data set from our previous study (see section 3 and Loconsole and Börstler (2005)). The models were then validated using a second new dataset. The data collection was semi-automatic, carried out by the authors by studying historical project’s documentation. We analysed the files starting from the first available revision, following the rules described in appendix A.

The decision to perform a correlational study was based on the widespread usage of this kind of study in the field. They are often the only option in industrial settings (Briand and Wüst, 2002).

Context of the study

We analysed and collected data from the requirements specifications of two different software projects performed at BAE Systems Hägglunds AB, Sweden (see table 1). The company produces automotive systems with embedded software and is ISO9001 and ISO14001 certified. At the time of the analysis the software systems had been in operation for approximately 24 months. The Rational Unified Process (RUP) was used in both projects.

Table

The goal for project A was to develop external diagnostics software for personal computers. This software system comprises fourteen small use case models (UCM) with a total of 39 use cases. According to the project terminology, a file containing a UCM is made up of an introduction, a revision table, a use case diagram, and a description of all actors and use cases (see Loconsole and Börstler (2005) for the UCM template utilised in project A). Use case modelling was the only technique used in this project for describing functional requirements. The vision document contained only sketchy, high level requirements and was not analysed. Seven non functional requirements were described in one additional file.

Project B developed an information and control system for the vehicles constructed by the company. This software system comprises 22 use cases and other non functional requirements described in two documents. According to the project terminology, a use case description contains the following sections: overview, revision history, references, description, state-diagram, normal flow, alternative flows, special requirements, start conditions, end conditions, and extension points. The actors of this system were described in a higher level requirements specification document called “use case summary”. In these projects, we consider use cases as requirements 3.

As can be observed from table 1, the documentation for projects A and B did not match completely, even though the projects were developed in the same company. No developer worked on both systems. The objects chosen for the study were the requirements documents of the two projects described above. In project A we analysed fourteen files, each containing a very small use case model. In project B we analysed twenty-two files, each containing one
use case. Other documentation, used to understand the projects A and B, were vision documents (the top level requirements specification), the use case summary (where we counted the actors), project plans, iteration plans, and test plans.

footnote: We are aware of the fact that some researchers do not consider use cases as requirements (Ham, 1998; Lau, 2004; Schneider and Winters, 1998; Young, 2004).

Dependent variables
To investigate the relationship between requirements size measures and requirements volatility, we had to choose a suitable and practical measure of volatility as the dependent variable of our study. Theoretical definitions of requirements volatility are presented in (Baumert and McWhinney, 1992; Nurmuliani et al., 2004; Raynus, 1999; Rosenberg and Hyatt, 1996), while operational definitions can be found in (Baumert and McWhinney, 1992; Chirsisis et al., 2003; Hyatt and Rosenberg, 1996; Loconsole and Böorstler, 2005; Nurmuliani et al., 2004; Raynus, 1999; Stark et al., 1999). Baumert and McWhinney (1992), suggest to measure source and state of change, while Nurmuliani et al. (2004) take into consideration the source of change in their theoretical definition of volatility. Except these two cases, all definitions have several things in common:

1. They express the changing nature of requirements during the software development.
2. They focus on the amount of changes (additions, deletions, and modifications) to requirements.
3. They do not consider the cause of change and the semantics of a change, i.e. in what way a change impacts development.

That means that volatility is treated as a quantitative measure. Likewise, we define requirements volatility as the total number of changes to a requirements document, and will measure it as the number of changes (NCHANGE) to a requirements document. There is one significant difference between our operational definition of volatility and the ones above. We look at volatility document by document instead of treating all requirements of a project as one entity. This makes it possible to distinguish requirements documents with high levels of volatility from those with low levels of volatility.

NCHANGE does not measure volatility directly. However, NCHANGE is a direct measure that can be used to calculate other measures, like change density and frequency of change. Change density is useful when we want to compare requirements between each other and say that a certain requirement is more volatile than another. Frequency of change can be obtained by monitoring is calculated by applying NCHANGE within specific time intervals. In this way we can also identify volatility trends.

Please note that we do not do any cause-effect or impact analyses of individual changes to requirements. Such qualitative analyses would require other types of measures, like for example the type of a change or the number of artefacts affected by a change. Such data is usually not available early on in the development. The downstream artefacts that could possibly be affected by a change (design and code for example) are not available yet.
Our dependent variable NCHANGE has been determined by comparing versions of requirements documents by means of a tool and counting the changes from one version of a document to the next. A detailed description of the counting rules can be found in appendix A.

Independent variables

The choice of the independent variables depends on the entity and the size of the systems measured. Because the projects under analysis are different from each other, it is necessary to select general measures that can be applied in both project contexts. This is also necessary to increase general applicability of results.

The entities analysed in the two projects were requirements documents. In project A we analysed fourteen files each containing a small use case model. In project B we analysed twenty-two files, each containing one use case. Intuitively, the larger the document the more changes there are. Therefore, we believe that the size of requirements is the most influential factor affecting volatility. The size measures “number of actors interacting with the use cases described in the file” (NACTOR), “number of lines per file” (NLINE), “number of words per file” (NWORD), “number of use cases per file” (NUC), and “number of revisions per file” (NREVISION) are the independent variables chosen for this study. As suggested by Fenton and Pfleeger (1996), size can be seen as composed of length, functionality, and complexity. In our case, NLINE and NWORD are measures of length, NACTOR and NREVISION are measures of complexity, and NUC is a measure of functionality. These measures are quite intuitive. The NLINE and NWORD are simply a count of lines and words of the files analysed and were calculated by the authors using a computerised tool. NACTOR is a count of the number of actors interacting with the use cases described in each file analysed. NREVISION is a count of the revisions for each file. A revision is a version of a file with a unique identifier.

The size of a requirement document can be computed at varying levels of granularity, because the requirements documents are organised hierarchically. We did not collect measures at higher or lower abstraction level, because we considered those requirements as either too vague or too close to the design level.

Selecting NLINE as independent variable might seem controversial. Like lines of code (LOC) as a size measure for program size, NLINE depends on the language used and formatting style. As pointed out by Armour (2004), what we actually want to measure is how much knowledge there is in our system or file. Unfortunately, there is not yet an empirical way to measure knowledge. Another possible choice for the independent variable could be use case points (UCPs) (Schneider and Winters, 1998). They have, for example, been used in effort estimation models (Anda, 2002). However, UCPs are not generally applicable. The definition of UCPs is based on a classification of use cases and a number of environmental factors (similar to the cost drivers in COCOMO (Boehm, 1981)). This information was not available for our projects. It is furthermore difficult, if not impossible, to classify use cases automatically. UCP measures can therefore not be collected automatically, which makes them unsuitable when non-intrusive measurement is required.
Further choices for independent variables could be measures like “number of associations between use cases” or “number of steps in scenarios”. However, such measures are dependent on the use case format used. If use cases are described in plain text only (the most basic format), this information is not available.

Volatility depends on many different factors, not only the size of requirements. It might for example be useful to include project factors, like the type of project or system, type of development process, team size or amount of communication or collaboration between stakeholders. Also factors related to the changes affect volatility, like the type, timing, and urgency of a change. Ideally one would wish to determine volatility using historical data representing all the factors above. However, since our concern is predicting volatility and not analysing effects of individual changes to requirements, such data is less important. In practice only the highly relevant variables for which reliable data are available are useful. Further variables may be incorporated at a later stage as understanding grows.

Hypothesis

The hypothesis of the study is the following: the size measures NACTOR, NUC, NWORD, NLINE, and NREVISION are good predictors of number of requirements changes. Our hypothesis is built on the idea that larger requirements are affected by changes more than smaller ones, because they contain more information. The relationship between number of changes and our five measures of size of a requirements document could be causal. However, in this study we only looked at whether a relationship exists, and whether our measures of size can be used as predictors of number of changes to requirements. The study does not make any claims with respect to causality which can be proven only by performing controlled experiments. In industrial environments it is usually hard to perform controlled experiment. For real projects, it is difficult to control variables, such as project length, the experience of customers and developers in specifying requirements, the techniques used to collect and document requirements, and the company’s maturity (in requirements and software processes in general). In academic environments, on the other hand, the small size of the projects does not usually allow to check for changes to requirements (Loconsole and Börstler, 2004).

6.5 Construction of prediction models

As described in section 3, in our previous study we found a strong correlation between four of the five size measures introduced in section 4.3 with total number of changes (Loconsole and Börstler, 2005). Furthermore, the scatter plots of our measures versus number of changes show approximately a linear correlation (see figure 1 and 2), especially for the length measures. Based on those results, we have analysed the ability of our measures to predict the number of changes to requirements. For this analysis we applied univariate and multivariate linear regression which is suggested to predict interval and ratio scale dependent variables (Briand and Wüst, 2002).
The data analysis is obtained by following the procedure suggested by Briand and W"ust (2002); Briand et al. (2000), which is also described in statistical books such as Draper and Smith (1966). We start with the descriptive statistics for projects A and B (section 5.1). The principal component analysis (5.2), univariate analysis (5.3), multivariate regression analysis (5.4), sanity tests on the regression models (5.5), and evaluation of goodness of fit (5.6) were performed only on data set A. In section 5.7 we evaluate the prediction models by applying them on data set B.

Descriptive statistics

Table 2 shows the descriptive statistics for data sets A and B. The columns SE Mean, StDev, and IQR state respectively mean standard error, standard deviation, and inter quartile range.

In both data sets, the mean NCHANGE is larger than the median 4. The standard deviation in data set B is higher than the mean, this is due to some outliers. Furthermore, the mean is more than two times higher than the mean in data set A. This implies that the prediction models, built based on data set A, will probably predict low numbers of changes for data set B.

footnote: Extremely high or low measurements will not affect the median as much as they affect the mean. Thus, when we deal with skewed populations we may prefer the median to the mean to express central tendency (Zar, 1999).

NACTOR, NUC, and NREVISION have relatively low means, standard deviations, and variances in both data sets. Low variance measures do not differentiate entities very well, therefore they are not likely to be useful predictors. The low variance of NACTOR and NUC is due to the fact that the requirements documents under analysis contain very few use cases and actors. However,

NACTOR and NUC can be expected to be low, since UCMs should be kept small according to common guidelines (Jacobson, 2004; Lilly, 1999). The totals of NACTOR in the two data sets are comparable, while the range and the median are very different. This means that the prediction models based on NACTOR will probably have low accuracy when applied to data set B.

In data set B, NUC has almost zero variance because in project B each file describes only one use case (except one file which described two use cases). According to Briand et al. (2000), only measures with more than five non zero data points should be considered for further analysis. This means that NUC might be a useless measure when we have one use case per file. In this case a better measure might be number of use cases per UCMs.

NWORD has the largest mean and standard deviation in data set A. The mean values of NWORD and NLINE in data set B are larger than the median, due probably to some outliers.
Besides some exceptions (NUC and NREVISION), the ranges of the measures in data set B are larger than in data set A. The totals are in the order of two times of data set A. In general, in data set B the measures have larger variation than in data set A. Except for NUC in data set B, our measures have all more than five non zero data points. There is therefore sufficient variance in all the measures to proceed with the analysis. The subsequent analysis will be done on data set A. After that, the constructed prediction models will be applied on data set B for evaluation.

Principal component analysis

Principal component (PC) analysis is performed in order to form a smaller number of uncorrelated variables, by selecting the independent variables with high loadings. For a set of n measures, there are at most n orthogonal PCs, which are calculated in decreasing order of variance they explain in the data set. Table 3 shows the results of the varimax rotation performed on data set A. This technique allows to identify a clearer pattern of loadings. For each PC, we provide its eigenvalue, the variance of the data set explained by the PC (in percent), and the cumulative variance. Absolute values above 0.7 are set in boldface.

As we can see in table 3, NLINE and NWORD get high factor loadings in PC1. They express the same orthogonal dimension, i.e. the length of requirements documents. NREVISION, NACTOR, and NUC get high loadings in PC2, PC3, and PC4, respectively. PC2 and PC3 express complexity, while PC4 expresses functionality dimensions. Among the five PCs shown in the table, three of them show sufficient variance, as shown by the scree plot in figure 3. The three PCs with high loadings capture 96.2% of the variance in the data set. PC4 and PC5 account for too small percentage of variance and should therefore be eliminated. Thus the measure NUC will not be considered in the multivariate analysis.

Univariate regression analysis

The results of the univariate analysis on data set A are shown in table 4. Regression analysis is conducted here to investigate the importance of each of the five size measures (the independent variables) in determining the number of changes (dependent variable). We test the hypothesis that the independent variables are significantly correlated with the dependent variable. This is also a way to screen out measures that are not likely to be significant predictors in multivariate models. We applied linear regression to each of the five measures (also called ordinary least squares regression), which is most suitable to predict a dependent variable at the interval and ratio scale (Briand and Wüst, 2002). Each of the measures was found to have a statistically significant positive relationship with NCHANGE. Table 4 summarises the results. For each measure, the regression coefficient, its standard error, the coefficient of determination (R2), and the statistical significance (P-value) are shown. The P-value is the
probability that the coefficient is different from zero by chance. Only measures that are significant at $\alpha = 0.05$ should be considered for the subsequent multivariate analysis (Briand and Wüst, 2002).

Model 2 has NUC as covariate, which had low variance in data set A (see table 2). This model has also the highest P-value and lowest R² (even though it is still significant). Similarly, model 5 has low R². It also has one influential outlier (the outlier is influential if the significance of the model changes when performing the test without the outlier). Therefore, this model will be evaluated without the outlier (as suggested by Briand and Wüst (2002)). A deeper analysis of outliers and model checking is done in section 5.5.

Observing table 4, the R² value of model 3 is 80.2. This means that model 3 explains 80.2% of the variation in number of changes. Similarly we can interpret the results for the other models. The goal of this test was to determine if each measure is a useful predictor of number of changes. Although the measures are all significant, NACTOR, NLINE, and NWORD seem to be better predictors than NUC and NREVISION.

table multivariate

**Multivariate regression analysis**

In this section we present the construction of prediction models built on data set A, with the goal of accurately predicting the number of changes to requirements. Because this study is exploratory, we do not know which independent variables should be included in the prediction models. Usually, a stepwise selection process is used (Briand and Wüst, 2002; Levine et al., 2001; Zar, 1999). A common method to reduce the number of independent variables is to use the results from the principal components analysis as filter, selecting only the variables with high loadings in the significant PCs. In our case only three PCs had sufficient variance, and this lead us to discard the measure NUC. We applied the multivariate linear regression starting with four variables. The statistics tool used for the analysis (Minitab) gives the possibility to check for collinearity. As expected, NLINE and NWORD were found collinear, therefore we continued the statistical analysis excluding NLINE.

Only two models with two covariates were found significant (see table 5). Both models have NWORD as one of the covariates. Model 6 (NWORD and NREVISION) had one influential outlier. Therefore, in the following sections, it will be evaluated without the outlier (as suggested by Briand and Wüst (2002)). A deeper analysis of outliers and model checking is done in section 5.5. We discarded the models with a P-value higher than 0.05 in the t-test.

**Sanity tests on the regression models**

One of the threats to conclusion validity is the violation of assumptions of statistical tests. To be valid, the models have to satisfy some hypothesis on the residuals. We followed the tests suggested in (Levine et al., 2001). The sanity checks were performed on five univariate models and two multivariate models. See figure 4 for an example of the plots generated for the analysis of the residuals.
The results of the model checking is listed below.

(1) Linear relationship between response and predictors. The "lack-of-fit-test", did not show any evidence of lack of fit for $p \geq 0.1$ in any of the seven models.

(2) Homogeneity of variance (the residuals have constant variance). The residuals versus fits plot did not reveal patterns in any case.

(3) Independence of residuals. The results of the Durbin-Watson test are shown in table 6. The values we obtained were higher than the upper bound in all cases, therefore we conclude that there is no autocorrelation, the residuals are independent.

(4) Normality of the residuals. The normal probability plot show an approximately linear pattern consistent with a normal distribution. In all cases, except for NREVISION, there is a normal distribution of residuals.

(5) No unusual observations or outliers. Minitab provides statistics tools which facilitates the detection of influential points such as leverages values, Cook’s distance, and adjusted difference (DFITS). From the results of these statistics, we found that model 1 has two unusual observations. Models 2, 3, 5, and 6 have one unusual observation while 4 and 7 have no outliers. We tested if the outliers found were influential because it is important that the conclusions drawn are not dependent on few outlying observations. Analysing the models without the outliers, we obtained that all the models were still significant at $\alpha = 0.05$.

After the analysis of the residuals, we decided to discard models 2 and 5 from the models’ evaluation because the residuals were not normally distributed and the models have the lowest performance. Furthermore, model 2 cannot be applied to data set B, due to the very low variance of NUC in data set B.

**Evaluating goodness of fit**

In this section, we evaluate the goodness of fit of three univariate models and two multivariate models obtained in the previous paragraphs. For prediction models constructed using the ordinary least square, the goodness of fit is measured with $R^2$. According to Briand and Wüst (2002), "while this measure allow, to some degree, for comparison of accuracy between studies, such measure is an abstract mathematical artifact that does not very well illustrate the potential benefits of using the prediction models for decision making". The measures suggested to evaluate the accuracy of the prediction models are the following: mean magnitude of relative error (MMRE), the threshold measure $\text{pred(n)}$, correctness, and completeness (Briand and Wüst, 2002). The magnitude of relative error is defined as a measure of discrepancy between the actual and the fitted values. A low MMRE generally indicates an accurate model, reliable systems have MRE between 10-30%. The MMRE can be strongly influenced by few very high relative error values. Therefore, we have included the median magnitude of relative error (MdMRE) in the evaluation.

Another measure of accuracy is $\text{pred(l)}$ which provides an indication of overall fit for a set of data points. $\text{Pred(l)}$ is based on the MRE values for each data pair and it is defined as the percentage of the data pairs with MRE $\leq l$. For example, $\text{pred(0.30)} = 0.43$ means that 43%
of the fitted values fall within 30% of their corresponding actual values. The higher the pred
values the more reliable is the model. Values of pred(0.25) above 0.7 mean that the system is
reliable, but such performance is difficult to get (DeLucia et al., 2005).

Table goodness of fit

As we can observe in table 7, the values of MMRE are between 17-71%. Models 3 and 6
have MMRE between 10 and 30 percent, this indicate that the models are reliable. In partic-
ular, model 6 has the lowest MMRE. Models 1 and 4 have higher MMRE but the MdMRE
is close to 30%. Observing the pred values, no one of our models reach a pred(0.25) = 0.7 as
suggested for reliable models. However, models 3 and 6 predict more than 50% of the cases
with a relative error less than 25%. Considering a relative error of 50%, models 3 and 6 can
predict more than 85% of the cases.

The ideal situation would be to compare the results obtained with other prediction models
of number of requirements changes, but we have not found such models. Considering other
research areas like object oriented design and software maintenance, the accuracy of our mod-
els is comparable with those presented in Genero et al. (2003) (whose best model has a
MMRE=0.24) and MacDonell (1997) (whose best model has a MMRE=0.21). It must be
noted that the prediction models in table 7 were applied to the same data set they were derived
from. We would therefore expect to get high accuracy. In the next section we will apply our
prediction models to data set B.

Validation of Prediction Models

Our situation is ideal for evaluating prediction models, since separate data sets are available
that have been derived from different projects, but within similar environments. The predic-
tion models are built from one data set and then used to make predictions for another project.
Project factors may affect the predictive power of a model and therefore it is important to val-
ify the model under conditions that as closely as possible resemble its usage conditions (Bri-
and and Wüst, 2002).

As we can observe from table 8, the values of MMRE are in the range 46-151%, when the
models are applied to data set B. These values are not similar to the recommended values for
reliable models. This is due to the differences between the data sets (see section 5.1). However,
the best models (3 and 4) have MMRE between 40 and 60%. Furthermore, model 3 predicts
more than 50% of the cases with a relative error less than 25%. This model performs better
than COCOMO (Boehm, 1981) (which has MMRE=0.6 and Pred(0.25)=0.27) and Jør-
gensen's best model (Jørgensen, 1995) (MMRE=1.0 and Pred(0.25)=0.26). Model 6 which
had the lowest MMRE in data set A does not perform very well on data set B, it seems to be
fitted for dataset A. Model 3 seems to be a stable model, it has a good performance on both
data sets.

Table validation of prediction model
It is interesting to note that the univariate models have higher accuracy compared to the multivariate models. Furthermore, the best models are those which depend on the length of the requirements and not on the functionality or complexity. This result is in accordance with the principal component analysis, where the PC1 was the most influential.

**Threats to validity**

A discussion of possible threats to validity will help us to qualify the results and highlight some of the issues associated with our study. A detailed list of possible threats is presented in (Wohlin et al., 2000).

**Conclusion validity.** One issue that could affect the conclusion validity is the relative small size of the sample data. Concerning data quality, the data for NLINE and NWORD have been collected using a computerised tool and are therefore reliable. The data collection for NCHANGE, NUC, NACTOR, and NREVISION involved human judgment. However, we have defined measurement rules to keep the judgment as objective as possible (see appendix A). Furthermore, the measurement rules have been tested; three subjects have independently applied the rules on two versions of a requirement document obtaining the same results. Finally, because the study has been done in retrospect, there is a risk that the data is imprecise, e.g., that the available files (use cases models and use cases) are incomplete. The risk of imprecise data is present even when practitioners capture data when they work. For retrospective studies this risk is even higher. However, this threat is minimal, the records were complete, the version handling system had no dangling references to missing documents.

**Construct validity.** The construct validity is the degree to which the variables are accurately measured by the measurement instruments used in the study. The construct validity of the measures used for the independent and dependent variables, is ensured by the theoretical validation performed in (Loconsole and Börstler, 2003).

**External validity.** Our threats to external validity are minimal because the study was performed in an industrial environment, therefore the materials used, and the projects themselves were real. Other factors that may affect the external validity of this study could be the project length, the experience of customers and developers in specifying requirements, the techniques used to collect and document requirements, and the company’s maturity (in requirements and software processes). The projects were developed in the same company and used (approximately) the same processes, but everything else was different: size, personnel, and type of project. That means, we can assume that the models are valid for a relatively large class of projects.

**Internal validity.** The study described here is a correlational study. We have shown that all measures are significantly correlated to number of requirements changes. However, this relationship does not imply a causal relationship. Only a controlled experiment, where the measures would be varied in a controlled manner (and the other factors would be held constant), could demonstrate causality, as we have discussed in section 4.4. One threat to internal validity is the potential inaccuracy of collected data on changes to requirements. Change requests were not available. Actual changes to requirements documents were determined by compar-
ing all revisions of the requirements documents analysed. Rules of measurements on how to count changes were defined (see appendix A). However, we had access to and analysed all revisions of the requirements documents analysed. Version and configuration management is handled seriously at the company. All records were complete (see conclusion validity).

6.6 Discussion and conclusions

In this paper, we have described a correlational study on number of requirements changes performed at BAE Systems Hagglunds AB, Sweden. We collected and analysed data of two historical projects in the company for five measures of size (NUC, NACTOR, NLINE, NWORD, NREVISION). Applying univariate and multivariate regression analysis, we built prediction models using data collected on a medium size software project. We then evaluated the model's accuracy on a second, slightly larger, project developed at the same company.

Our data analysis shows that all measures have a positive correlation with NCHANGE, i.e. we can accept our hypothesis that the size measures NACTOR, NUC, NWORD, NLINE, and NREVISION are good predictors of number of requirements changes.

The measures are accurate predictors, except NUC and NREVISION that were associated to a low coefficient of determination. The accuracy of the measures as predictors is also shown by the low values of MMREs and high values of pred (see table 7). Looking at the coefficient of determination of the univariate models, the best predictor seems to be NLINE, followed by NWORD, NACTOR, NREVISION, and NUC. Similar results were obtained by measuring the MMREs (see table 7). The multivariate model 6, which combines the covariates NWORD and NREVISION, shows the best performance on data set A. It has the highest coefficient of determination, lowest MMRE, and highest Pred(0.25), but has weak performance on data set B. This may be due to the differences between the two data sets especially because NREVISION had different means in the two data sets (see table 2). When applying the models on data set B, NLINE was the best predictor, followed by NWORD, the multivariate model with covariates NWORD and NACTOR, and the univariate model NACTOR. The predictive ability of NLINE is comparable to those of common prediction models from other areas, such as COCOMO (Boehm, 1981) for effort estimation. It is interesting to note that our best models are those depending only on the length of the requirements documents and not on the functionality or complexity or a combination thereof (see principle component PC1 in table 3.

Please note that our dependent variable NCHANGE does not measure volatility directly. However, NCHANGE is a basic measure that can be used to easily compute other common (volatility) measures, like change density or change frequency (see section 4.2).

By regularly comparing the actual number of changes to the current value of NCHANGE, project managers can identify critical requirements and allocate resources for a closer analysis of the reasons for their volatility. In this way they can minimise the risks of schedule and cost overruns.
In the present work, we deal only with one factor of volatility; the number of changes to requirements. As discussed in sections 4.2 and 4.3, volatility is a quite complex concept depending on many more factors than size. For a deeper analysis of volatility it is suggested to investigate qualitative aspects such as why the changes occur, how critical the changes are, the type and phase where the changes occur. Investigating many qualitative aspects on many requirements documents is expensive, subjective, and not feasible. However, studying the impact of a change might help to “classify” changes and to identify the most critical changes. In a qualitative analysis, we would care only about important changes since all others will not affect the project much. With our prediction models we can quantify the “instability” of requirements in order to identify the critical ones. When the critical requirements are identified, we can perform a deeper analysis of the changes in order to figure out the problems with the requirement.

Our approach is based on use case based requirements. However, the defined measures are quite general and do not depend on a specific use case format or template. They can be used for all kinds of use case documents written in textual form. Furthermore, the measures NLINE and NWORD could be applied to any kind of text-based requirements document. Our approach of using counts is simple, effective, easily interpreted and can be completely automated.

We believe it is possible to develop prediction systems based upon simple measures such as ours. The reason for our confidence is that the correlational study presented here is based on two different industrial projects. Although the exact nature of the prediction systems will vary from company to company, the underlying principle is the same. That is, developers can collect simple measures derived from requirements documents, and build effective prediction systems using techniques like linear regression analysis. Models must however be calibrated to suit different environments (MacDonell, 1997).

6.7 References


6.8 Measurement rules

In this section, we describe the rules we adopted for measuring the requirements documents of the two analysed projects. For this study we considered files as units of requirements documents. We had full access to the repository of the projects and retrieved all existing revisions of all files analysed from the start of the projects until the point in time of the analysis.

The measure NCHANGE was obtained by counting the number of changes to a unit of requirement document. The size of change is usually dependent on the effort spent for the change or the number of artifacts impacted by the change. Unfortunately, this information was not available in the current projects. Therefore, the size of change was estimated by the authors. We compared two successive versions of a file using the ‘track changes: compare documents’ tool in MS Word. Each word added or deleted was considered as one change, while each word substituted was considered as two changes (one deletion plus one addition). Furthermore, adding a picture, resizing, adding a frame, adding or deleting a detail in a picture were all considered as one change. Intuitively, adding a picture should be considered a bigger change compared to resizing it. However, if changes are made in the pictures these are usually mirrored with changes in the text.
Exceptions to these rules were the following. We did not count addition or deletion of empty space, empty lines added or deleted, page breaks or tabs without text. Automatic changes like the date in the headers and the filename in the footers were not considered changes. If a section was inserted or deleted and the successive section's numbers were changed, they were not considered as changes. The same change repeated many times in different paragraphs was considered as many changes. There were changes like a substitution of an newline where a word appears in red. We did not considered these words as changed.
A correlational study on four size measures as predictors of requirements volatility

Annabella Loconsole, Jürgen Börsler

Abstract

In this paper, we present a correlational study with the goal of predicting requirements volatility for a medium size software project. Based on the data collected from two industrial software projects for four measures of size of requirements (number of actors, use cases, words, and lines), we have built and evaluated prediction models for requirements volatility. Performing a cross systems validation, the best model showed a MMRE = 0.25, which can be considered reliable. Although our models are likely to have only local validity, the general method for constructing the prediction models could be applied in any software development company.
7.1 Introduction

Developing requirements is said to be a learning rather than a gathering process [35]. As a consequence, requirements change frequently, even during later stages of the development process. Software requirements that change often are usually said to be volatile. Studies show that requirements volatility has a high impact on project performance [33, 39, 43]. Since we cannot expect requirements to be stable, even when requirements engineering tasks are performed well, we should at least carefully monitor and control them throughout the software life cycle.

In this paper, we describe a correlational study with the goal of empirically validating four measures of size as predictors of requirements volatility. We built four prediction models using data collected for a medium-size software project developed at BAE Systems Hagglunds AB, Sweden. We then evaluated the accuracy of one model by applying it on a set of data collected for a second project at the same company. The results show that the length measure “number of lines” (of a requirements document) is a good predictor of volatility. Other measures of size (number of actors, number of use cases, number of words) were not found to be significant predictors.

Our work is unique regarding two aspects. First, we aim to predict requirements volatility, while volatility is usually chosen as an independent variable i.e. as predictor of other software or project attributes, like for example in [1, 17, 20, 39]. Second, we are concerned with volatility of smaller units of requirements, instead of treating volatility as a property of the whole set of requirements of a project.

The remaining part of the paper proceeds as follows: section 2 describes the research related to empirical validation of measures in general and work related to requirements volatility. In section 3 we briefly summarise our previous empirical studies, which investigated the relationship between measures of size of requirements and number of changes to requirements. Section 4 describes the goals, hypotheses, and data collected in the present correlational study. The construction of the prediction models is described in section 5 and the validation is presented in section 6. The threats to validity are presented in section 7. Finally, discussions and conclusions are presented in section 8.

7.2 Related work

Although requirements volatility is a well-studied area, there is relatively little empirical research. The empirical research available is mostly concerned with the impact of requirements volatility on other software or project attributes, like for example project performance or risk [1, 43], software maintenance [39], and defect density [20, 31].

Correlational studies are often used to empirically validate software measures. The reason for validating measures is to demonstrate their practical utility, i.e. to show that there is a consistent relationship between the measure and an external attribute [14, 22, 38, 44]. It is important to ensure that the data collected for a certain measure is related to the actual property investigated (the attribute to be measured). Otherwise time and money is spent for collecting
useless data. To our knowledge, only [28, 29] describe an empirical validation of requirements volatility measures performed in an industrial setting. In [28], we proved that there is a high correlation between four measures of size and number of changes to use case models (see section 3 for further details). In [29] prediction models of number of changes to requirements were constructed and validated.

Other approaches that can support a predictive view of requirements stability are described by Bush and Finkelstein [8, 9]. Starting from an initial set of requirements, they describe a process that helps to create worlds of possible evolutions of requirements. They also report on positive results from an industrial case study validating the approach. However, using their process is more complex and time consuming than ours, which is based on use case requirements, and is simpler and better suited for small and medium sized companies. Nevertheless, except for the use case diagrams, the requirements are mainly text based. Therefore, it seems possible to generalise our results. Further studies are, however, needed to support such a claim.

7.3 Background
In two industrial empirical studies [28, 29], we investigated measures of volatility for a medium size software project. In the first study [28], our goals were: (1) to empirically validate a set of measures associated with the volatility of requirements documents; and (2) to investigate the correlation between perceived and measured volatility. We collected size and change data in retrospect for all versions of requirements documents of the software project. In addition, we determined the perceived volatility by interviewing stakeholders of the project.

The spearman correlation coefficient was calculated between each measure of size of requirements documents and the size of change to requirements documents. Requirements in the project were described in terms of use cases and each requirements document contained one use case model. The size of a requirements document was measured in terms of number of lines, number of words, number of use cases, and number of actors.

The data analysis showed a high correlation between each of the size measures and the total number of changes. This suggests that our measures of size of requirements documents are good indicators of the number of changes for use case based requirements documents.

For the second goal above, we could not find significant correlations between any of our four volatility measures and the rating of volatility by the experts. This implies that the developers’ perceptions of number of changes were not good indicators of requirements volatility for the project analysed.

Based on these results, we performed a correlational study [29] with the goal of evaluating the best predictor of number of changes. We built and evaluated prediction models for number of changes to requirements. For our best model we calculated a pred(0.25)=0.5, which is better than the accuracy of common effort prediction models like for example CO-COMO [5]. These results suggest that managers at this company would benefit from measuring their projects because of the risk to take wrong decisions based solely on their own and the developers perceptions.
7.4 Description of the correlational study

The goal of the present study is to analyse the ability of four specific measures to predict the volatility of requirements, using two data sets. Prediction models were constructed applying linear regression analysis using the data set from project A (see table 1), used also in our previous study [28]. The best model was then validated using a second dataset from project B. The data collection was semi-automatic, carried out by the authors by studying the documentation of the historical projects A and B. From the first available revisions of requirements documents, all files were analysed following the rules described in appendix A.

The decision to perform a correlational study was based on the widespread usage of this kind of study in the field. They are often the only option in industrial settings [6].

Context of the study

We analysed and collected data from the use case-based requirements specifications of two different software projects performed at BAE Systems H¨agglunds AB, Sweden (see table 1). The company produces automotive systems with embedded software and is ISO9001 and ISO14001 certified. At the time of the analysis the software systems had been in operation for approximately 24 months. The Rational Unified Process (RUP) was used in both projects. The goal for project A was to develop external diagnostics software for personal computers. This software system comprises fourteen small use case models (UCM) with a total of 39 use cases. According to the project terminology, a file containing a UCM is made up of an introduction, a revision table, a use case diagram, and a description of all actors and use cases (see [28] for the UCM template utilised in project A). Use case modelling was the only technique used in this project for describing functional requirements. The vision document contained only sketchy, high level requirements and was not analysed. Seven non functional requirements were described in one additional file.

TABLE 11: Key data of the two projects analysed

<table>
<thead>
<tr>
<th></th>
<th>Project A</th>
<th>Project B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of developers</td>
<td>10 (incl. 2 proj. managers)</td>
<td>15</td>
</tr>
<tr>
<td>Project duration</td>
<td>30 months</td>
<td>48 months</td>
</tr>
<tr>
<td>Use case documents (UCD)</td>
<td>14 files</td>
<td>22 files</td>
</tr>
<tr>
<td>Average UCD size in kB (Median)</td>
<td>60</td>
<td>76.5</td>
</tr>
<tr>
<td>Total number of req. documents</td>
<td>16</td>
<td>26</td>
</tr>
<tr>
<td>Vision/overview documents</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Supplementary req. documents Total number of use cases</td>
<td>1</td>
<td>22</td>
</tr>
</tbody>
</table>

footnote: We are aware of the fact that some researchers do not consider use cases to be requirements [16, 23, 37, 41].

Project B developed an information and control system for the vehicles constructed by the company. This software system comprises 22 use cases and other non functional requirements described in two documents. According to the project terminology, a use case description
contains the following sections: overview, revision history, references, description, state-diagram, normal flow, alternative flows, special requirements, start conditions, end conditions, and extension points. The actors of this system were described in a higher level requirements specification document called “use case summary”.

As can be observed from table 1, the documentation for projects A and B did not match completely, even though the projects were developed in the same company. No developer worked on both systems. The objects chosen for the study were the requirements documents of the two projects described above. In project A we analysed fourteen files, each containing a very small use case model. In project B we analysed twenty-two files, each containing one use case. Other documentation, used to understand the projects A and B, were vision documents (the top level requirements specification), the use case summary (where we counted the actors), project plans, iteration plans, and test plans.

Independent variables
The choice of the independent variables depends on the entity and the size of the systems measured. Because the two projects under analysis are different from each other, it is necessary to select general measures that can be applied in both project contexts. This is also necessary in order to increase the general applicability of results.

The entities analysed in the two projects were requirements documents. Intuitively, the larger the document the more changes there are. Therefore, we believe that the size of requirements is the most influential factor affecting volatility. The size measures “number of actors interacting with the use cases described in the file” (NACTOR), “number of lines per file” (NLINE), “number of words per file” (NWORD), and “number of use cases per file” (NUC), are the independent variables chosen for this study. As suggested by [14], size can be seen as composed of length, functionality, and complexity. In our case, NLINE and NWORD are measures of length, NACTOR is a measure of complexity, and NUC is a measure of functionality. These measures were calculated by the authors using a computerized tool.

The size of a requirement document can be computed at varying levels of granularity, because the requirements documents are organised hierarchically. We did not collect measures at higher or lower abstraction levels, because we considered those requirements as either too vague or too close to the design level.

Selecting NLINE as independent variable might seem controversial. Like lines of code (LOC) as a size measure for program size, NLINE depends on the language used and formatting style. As pointed out by [3], what we actually want to measure is how much knowledge there is in our system or file. Unfortunately, there is no empirical way to measure knowledge. A possible choice for the independent variable could be use case points (UCPs) [2, 37]. However, UCPs are not generally applicable. The definition of UCPs is based on a classification of use cases and a number of environmental factors (similar to the cost drivers in COCOMO [5]). This information was not available for our projects. Furthermore, the classification of use cases is a subjective activity, therefore it is not possible to collect UCPs automatically.
Further choices for independent variables could be measures like “number of associations between use cases”, “number of steps in scenarios”. However, such measures are dependent on the use case format used. If use cases are described in plain text only (the most basic format), this information might not be available. Therefore, we discarded such measures for the sake of generality.

Volatility depends on many different factors, not only the size of requirements. It might be useful to include project factors (like the type of project or system, type of development process, team size or amount of communication or collaboration between stakeholders) among the independent variables. Also factors related to the changes affect volatility, like the type, timing, and urgency of a change. Ideally one would wish to determine volatility using historical data representing all the factors above. However, since our concern is predicting volatility and not analysing effects of individual changes to requirements, such data is less important. In practice, only the highly relevant variables for which reliable data are available are useful. Further variables may be incorporated at a later stage as understanding grows.

Dependent variables
Our goal was to investigate the relationship between requirements size measures and requirements volatility. We therefore had to choose a suitable and practical measure of volatility as the dependent variable of our study. Theoretical definitions of requirements volatility are presented in [4, 32, 34, 36], while operational definitions can be found in [4, 10, 18, 28, 32, 34, 39]. Baumert and McWhinney [4], suggest to measure source and state of change, while Nurmuliani et al. [32] take into consideration the source of change in their theoretical definition of volatility. Except these two cases, all definitions have several things in common: 1. they express the changing nature of requirements during the software development; 2. they focus on the amount of changes (additions, deletions, and modifications) to requirements; 3. they do not consider the cause of change and the semantics of a change, i.e. in what way a change impacts development. That means that volatility is treated as a quantitative measure. Likewise, we define requirements volatility as the amount of changes to a requirements document over time and we measure it as the sum of the change densities of a requirements document in time.

Volatility=

Our operational definition of requirements volatility is a function of: number of changes (NCHANGE), time measured in number of revisions (NREVISION), and size of the requirements document measured in NWORD. NCHANGE is a count of changed words, therefore, NWORD was chosen to calculate the change density (having the same unit of measurement). NREVISION is a count of all the revisions for each file. A revision is a version of a file with a unique identifier.

There is one difference between our operational definition of volatility and the ones described earlier above. We look at volatility document by document instead of threat when all requirements are one set. This makes it possible to distinguish requirements documents with high levels of volatility from those with low levels of volatility. Please note that we do not per-
form any cause-effect or impact analysis of individual changes to requirements. Such qualitative analysis would require other types of measures, like for example the type of a change or the number of artefacts affected by a change. Such data is usually not available early on in the development.

Our dependent variable has been determined by comparing versions of requirements documents by means of a tool and counting the changes from one version of a document to the next. A detailed description of the counting rules can be found in appendix A.

Hypothesis
The hypothesis of the study is the following: the size measures NACTOR, NUC, NWORD, and NLINE are good predictors of requirements volatility. Our hypothesis is built on the idea that larger requirements are affected by changes more than smaller ones, because they contain more information. The relationship between volatility and our four size measures of a requirements document could be causal. However, in this study we only looked at whether a relationship exists, and whether our measures of size can be used as predictors of requirements volatility. The study does not make any claims with respect to causality which can be proven by performing controlled experiments [40]. In industrial environments it is usually hard to perform controlled experiments. For industrial projects, it is difficult to control variables, such as project length, the experience of customers and developers in specifying requirements, the techniques used to collect and document requirements, and the company's maturity (in requirements and software processes in general). In academic environments, on the other hand, the small size of the projects does not usually allow to check for changes to requirements [27].

7.5 Construction of prediction models
As described in section 3, in our previous study we found a strong correlation between the four size measures introduced in section 4.2 with total number of changes [28, 29].

Based on those results, we have analysed the ability of our measures to predict the volatility of requirements. For this analysis we applied linear regression which is suggested to predict interval and ratio scale dependent variables [6].

The data analysis is obtained by following the procedure suggested by Briand et al. [6, 7], which is also described in statistics books such as Draper and Smith [13]. Briand et al. provide constructive guidelines to facilitate the work of empirical validation of measures. We start with the descriptive statistics performed on data sets A and B (section 5.1).

The principal component analysis (5.2), univariate regression analysis (5.3), multivariate regression analysis (5.4), sanity tests on the regression models (5.5), and evaluation of goodness of fit (5.6) were performed only on data set A. In section 6 we validate one prediction model by applying it on data set B. The choice of the modelling technique for the regression analysis is mostly driven by the nature of the dependent variable: its distribution, scale, and whether it is discrete or continuous.
Descriptive statistics

Table 2 shows the descriptive statistics for data sets A and B. The columns SE Mean, StDev, and IQR state respectively mean standard error, standard deviation, and inter quartile range. In both data sets, the mean of NCHANGE and of Volatility is larger than the median, especially in data set B.

Footnote 2: Extremely high or low measurements will not affect the median as much as they affect the mean. Thus, when we deal with skewed populations.

TABLE 12: Descriptive statistics for data sets A and B

<table>
<thead>
<tr>
<th>Data set</th>
<th>Measures</th>
<th>Range</th>
<th>Total</th>
<th>Mean</th>
<th>SE Mean</th>
<th>StDev</th>
<th>Variance</th>
<th>Median</th>
<th>IQR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (14 files)</td>
<td>NACTOR</td>
<td>2</td>
<td>5</td>
<td>1.5</td>
<td>0.174</td>
<td>0.65</td>
<td>0.423</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>NUC</td>
<td>5</td>
<td>39</td>
<td>2.786</td>
<td>0.447</td>
<td>1.672</td>
<td>2.797</td>
<td>3</td>
<td>2.25</td>
</tr>
<tr>
<td></td>
<td>NLINE</td>
<td>105</td>
<td>1436</td>
<td>102.57</td>
<td>8.35</td>
<td>31.24</td>
<td>975.96</td>
<td>100.5</td>
<td>52.5</td>
</tr>
<tr>
<td></td>
<td>NWORD</td>
<td>852</td>
<td>8771</td>
<td>626.5</td>
<td>73.9</td>
<td>276.4</td>
<td>76375.5</td>
<td>663.5</td>
<td>492.8</td>
</tr>
<tr>
<td></td>
<td>NREV</td>
<td>5</td>
<td>89</td>
<td>6.357</td>
<td>0.44</td>
<td>1.646</td>
<td>2.709</td>
<td>6.5</td>
<td>3.25</td>
</tr>
<tr>
<td></td>
<td>NCHANGE</td>
<td>1109</td>
<td>10811</td>
<td>772.2</td>
<td>98.5</td>
<td>368.4</td>
<td>135738.2</td>
<td>754.5</td>
<td>658.5</td>
</tr>
<tr>
<td></td>
<td>Volatility</td>
<td>1.092</td>
<td>23.85</td>
<td>1.703</td>
<td>0.094</td>
<td>0.351</td>
<td>0.12</td>
<td>1.6</td>
<td>0.6355</td>
</tr>
<tr>
<td>B (22 files)</td>
<td>NACTOR</td>
<td>5</td>
<td>6</td>
<td>3.182</td>
<td>0.276</td>
<td>1.296</td>
<td>1.68</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>NUC</td>
<td>1</td>
<td>23</td>
<td>1.0455</td>
<td>0.0455</td>
<td>0.2132</td>
<td>0.0455</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>NLINE</td>
<td>378</td>
<td>2848</td>
<td>129.5</td>
<td>18.7</td>
<td>87.9</td>
<td>7720.1</td>
<td>109</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>NWORD</td>
<td>2379</td>
<td>16831</td>
<td>765</td>
<td>123</td>
<td>578</td>
<td>333794</td>
<td>606</td>
<td>822</td>
</tr>
<tr>
<td></td>
<td>NREV</td>
<td>9</td>
<td>67</td>
<td>3.045</td>
<td>0.434</td>
<td>2.035</td>
<td>4.141</td>
<td>2</td>
<td>1.25</td>
</tr>
<tr>
<td></td>
<td>NCHANGE</td>
<td>4465</td>
<td>26086</td>
<td>1186</td>
<td>240</td>
<td>1125</td>
<td>1266045</td>
<td>627</td>
<td>1183</td>
</tr>
<tr>
<td></td>
<td>Volatility</td>
<td>3.075</td>
<td>46.493</td>
<td>2.113</td>
<td>0.158</td>
<td>0.740</td>
<td>0.547</td>
<td>1.876</td>
<td>0.394</td>
</tr>
</tbody>
</table>

Therefore, the mean of NCHANGE in dataset B is higher than the mean in data set A. This implies that the prediction models, built based on data set A, will probably predict low volatility for data set B. NREV has relatively low means, standard deviations, and variances in both data sets. NCHANGE has the largest mean and standard deviation in data sets A and B. NACTOR and NUC have relatively low means, standard deviations, and variances in both data sets. The low variance of NACTOR and NUC is due to the fact that the requirements documents under analysis contain very few use cases and actors. However, NACTOR and NUC can be expected to be low, since UCMs should be kept small according to common guidelines [19, 25]. The totals of NACTOR in the two data sets are comparable, while the range and the median are very different. This means that the prediction models based on NACTOR would probably have low accuracy when applied to data set B. In data set B, NUC has almost zero variance because in project B each file describes only one use case (except one file which described two use cases). The mean values of NWORD and NLINE in data set B are larger than the median, due to some outliers.

Except for the measure NUC, the ranges of the measures in data set B are larger than in data set A. The totals are in the order of two times of data set A for the measures NLINE, NWORD, NCHANGE and Volatility. In general, in data set B the measures have larger var-
iation than in data set A. Except for NUC in data set B, our measures have all more than five non-zero data points. There is therefore sufficient variance in all the measures to proceed with the analysis [7].

The subsequent analysis will be done on data set A. After that, the constructed prediction model will be applied on data set B for evaluation. We may prefer the median to the mean to express central tendency [42].

Principal component analysis
Table 3 shows the results of the varimax rotation performed on data set A. For each principal component (PC), we provide its eigenvalue, the variance of the data set explained by the PC (in percent), and the cumulative variance. Absolute values above 0.7 are set in boldface.

TABLE 13: Rotated components for data set A

<table>
<thead>
<tr>
<th>Measures</th>
<th>PC1</th>
<th>PC2</th>
<th>PC3</th>
<th>PC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eigenvalue</td>
<td>3.0832</td>
<td>0.6318</td>
<td>0.2543</td>
<td>0.0308</td>
</tr>
<tr>
<td>Percent</td>
<td>0.771</td>
<td>0.158</td>
<td>0.064</td>
<td>0.008</td>
</tr>
<tr>
<td>Cumulative</td>
<td>0.771</td>
<td>0.929</td>
<td>0.992</td>
<td>1.000</td>
</tr>
<tr>
<td>NACTOR</td>
<td>0.270</td>
<td>0.150</td>
<td>-0.951</td>
<td>-0.017</td>
</tr>
<tr>
<td>NLINE</td>
<td>0.731</td>
<td>0.485</td>
<td>-0.437</td>
<td>-0.198</td>
</tr>
<tr>
<td>NUC</td>
<td>0.386</td>
<td>0.909</td>
<td>-0.156</td>
<td>-0.016</td>
</tr>
<tr>
<td>NWORD</td>
<td>0.882</td>
<td>0.380</td>
<td>-0.277</td>
<td>0.042</td>
</tr>
</tbody>
</table>

As we can see in table 3, NLINE and NWORD get high factor loadings in PC1. They express the same orthogonal dimension, i.e. the length of requirements documents. NUC and NACTOR get high loadings in PC2 and PC3, respectively. PC2 expresses functionality, PC3 complexity, and PC4 expresses other factors. Among the four PCs in table 3, two of them show sufficient variance, as can be seen in the scree plot in figure 1. The two PCs with high loadings capture 93% of the variance in the data set. PC3 and PC4 account for too small percentage of variance and should therefore be eliminated. Thus the measure NACTOR will not be considered in the multivariate analysis.

Univariate regression analysis
The results of the univariate analysis on data set A are shown in table 4. Regression analysis is conducted here to test the hypothesis that the independent variables are significantly correlated with the dependent variable. This is also a way to screen out measures that are not likely to be significant predictors in multivariate models. We applied linear regression to each of the four measures (also called ordinary least squares regression), which is most suitable to predict a dependent variable at the interval and ratio scale [6]. Two of the four measures had a statistically significant positive relationship with the dependent variable volatility.
Table 4 summarises the results. For each measure, the regression coefficient, its standard error, the coefficient of determination (R^2), and the statistical significance (P-value) are shown. The P-value is the probability that the coefficient is different from zero by chance. Only measures that are significant at $\alpha = 0.05$ should be considered for the subsequent multivariate analysis [6].

Observing table 4, models 2 and 4 (with covariate NUC and NWORD) have a P-value higher than the $\alpha = 0.05$, while models 1 and 3 are significant. Observing table 4, the R^2 value of model 3 is 40.3. This means that model 3 explains 40.3% of the variation of the dependent variable. Similarly we can interpret the results for the other model. The goal of this test was to determine if each measure is a useful predictor of volatility. In our case, the measures NACTOR and NLINE are significant predictors.

### TABLE 14: Univariate regression analysis for data set A ($\alpha = 0.05$)

<table>
<thead>
<tr>
<th>Models</th>
<th>Measure</th>
<th>Coeff</th>
<th>Std. Err</th>
<th>P-value</th>
<th>R2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NACTOR</td>
<td>0.3258</td>
<td>0.1241</td>
<td>0.022</td>
<td>36.5</td>
</tr>
<tr>
<td>2</td>
<td>NUC</td>
<td>0.08278</td>
<td>0.05566</td>
<td>0.163</td>
<td>15.6</td>
</tr>
<tr>
<td>3</td>
<td>NLINE</td>
<td>0.007133</td>
<td>0.002504</td>
<td>0.015</td>
<td>40.3</td>
</tr>
<tr>
<td>4</td>
<td>NWORD</td>
<td>0.0005351</td>
<td>0.0003324</td>
<td>0.133</td>
<td>17.8</td>
</tr>
</tbody>
</table>

Table 4 summarises the results. For each measure, the regression coefficient, its standard error, the coefficient of determination (R^2), and the statistical significance (P-value) are shown. The P-value is the probability that the coefficient is different from zero by chance. Only measures that are significant at $\alpha = 0.05$ should be considered for the subsequent multivariate analysis [6].

Observing table 4, models 2 and 4 (with covariate NUC and NWORD) have a P-value higher than the $\alpha = 0.05$, while models 1 and 3 are significant. Observing table 4, the R^2 value of model 3 is 40.3. This means that model 3 explains 40.3% of the variation of the dependent variable. Similarly we can interpret the results for the other model. The goal of this test was to determine if each measure is a useful predictor of volatility. In our case, the measures NACTOR and NLINE are significant predictors.
Multivariate regression analysis
The multivariate regression analysis was not applicable because from the PC analysis we discarded NACTOR and from the univariate analysis we discarded NWORD and NUC. This left us with only one independent variable.

Sanity tests on the regression models
One of the threats to conclusion validity is the violation of assumptions of statistical tests. To be valid, the models have to satisfy some hypotheses on the residuals. We followed the tests suggested in [24]. The sanity checks were performed on the univariate models 1 and 3. Figure 2 shows the plots generated by the analysis of the residuals. The results of the model checking is listed below.

![Residual Plots for Volatility vs NACTOR](image)

FIGURE 18, residual analysis
Linear relationship between response and predictors. The “lack-of-fit-test”, did not show any evidence of lack of fit in both models for \( p \leq 0.1 \) (which is a standard threshold).

2. Homogeneity of variance (the residuals have constant variance). The residuals versus fits plot did not reveal patterns in any case (see figure 2).

3. Independence of residuals. The results of the Durbin-Watson test are shown in table 5. The values obtained were higher than the upper bound in all cases, therefore we conclude that there is no autocorrelation; the residuals are independent.

4. Normality of the residuals. The normal probability plot showed an approximately linear pattern consistent with a normal distribution of the residuals for model 3. The residuals were not normally distributed in case of model 1, as can be seen in figure 2.

No unusual observations or outliers. The statistics used for the detection of influential points were leverages values, Cook’s distance, and adjusted difference (DFITS). From the results of these statistics, we found that model 3 did not have unusual observations, while two outliers were found for model 1. We tested if the outliers were influential because it is impor-

---

**TABLE 15**: Durbin-Watson test (sample size=14)

<table>
<thead>
<tr>
<th>Models</th>
<th>Measure</th>
<th>DW values</th>
<th>Critical values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NACTOR</td>
<td>2.18504</td>
<td>[1.04495, 1.35027]</td>
</tr>
<tr>
<td>3</td>
<td>NLINE</td>
<td>2.3664</td>
<td>[1.04495, 1.35027]</td>
</tr>
</tbody>
</table>

**FIGURE 19.** Residual analysis
tant that the conclusions drawn are not dependent on few outlying observations. The univariate regression model obtained by removing the two outliers from model 1 was not significant at $\alpha = 0.05$ ($P=0.07$), i.e. the outliers were influential.

After the models' checking, we decided to discard the model having NACTOR as covariate from the models' evaluation because the outliers were influential (the significance of the regression model changed).

**Evaluating goodness of fit**

In this section, we evaluate the goodness of fit of model 3 obtained in the previous paragraphs. The measures we used to evaluate the accuracy of the prediction models are the mean magnitude of relative error (MMRE) and the threshold measure pred(n). The magnitude of relative error is defined as a measure of discrepancy between the actual and the fitted values. Reliable prediction systems have MRE values between 10-30%. The MMRE can be strongly influenced by few very high relative error values. Therefore, we have included the median magnitude of relative error (MdMRE) in the evaluation. Another measure of accuracy is pred(n) which provides an indication of overall fit for a set of data points. Pred(n) is based on the MRE values for each data pair and it is defined as the percentage of the data pairs with MRE$n$. The higher the pred values the more reliable is the model. Values of pred(0.25) above 0.7 mean that the system is reliable, but such performance is difficult to get [12].

Observing the pred value in table 6, our model predicts more than 90% of the cases with a relative error less than 25%. Considering a relative error of 40%, the model predicts 100% of the cases. The ideal situation would be to compare the results obtained with other prediction models of volatility, but we have not found such models. Considering other research areas like object-oriented design and software maintenance the goodness of fit of our model is better than those presented by Genero et al. [15] (whose best model has a MMRE=0.24) and by MacDonnel [30] (whose best model has a MMRE=0.21). Similarly to our case, these models were evaluated on the same set they were constructed. We would therefore expect to get high accuracy. In the next section we will apply our prediction model to data set B.

**TABLE 16:** Evaluation of the goodness of fit (data set A)

<table>
<thead>
<tr>
<th>Models</th>
<th>Measure</th>
<th>MMRE</th>
<th>MdMRE</th>
<th>Pred(0.25)</th>
<th>Pred(0.4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>NLINE</td>
<td>0.14</td>
<td>0.1255</td>
<td>0.93</td>
<td>1</td>
</tr>
</tbody>
</table>

**TABLE 17:** Prediction model validation (data set B)

<table>
<thead>
<tr>
<th>Models</th>
<th>Measure</th>
<th>MMRE</th>
<th>MdMRE</th>
<th>Pred(0.25)</th>
<th>Pred(0.4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>NLINE</td>
<td>0.25</td>
<td>0.21</td>
<td>0.68</td>
<td>0.86</td>
</tr>
</tbody>
</table>
7.6 Prediction model validation

Our situation is ideal for evaluating prediction models, since separate data sets are available that have been derived from different projects, but within similar environments. The prediction model is built from one data set and then used to make predictions for another project. Project factors may affect the predictive power of a model and therefore it is important to validate the model under conditions that as closely as possible resemble its usage conditions [6].

The values of MMRE and of MdMRE when model 3 is applied to data set B, are shown in table 7. These values are in the range of the recommended values for reliable models. The model predicts 68% of the cases with a relative error less than 25% and it predicts 86% of the cases with a relative error less than 40%. This model performs better than COCOMO [5] (which has MMRE=0.6 and Pred(0.25)=0.27) and Jørgensen’s best model [21] (MMRE=1.0 and Pred(0.25)=0.26).

It is interesting to note that our model depends on the length of the requirements and not on the functionality or complexity. This result is in accordance with the principal component analysis, where the PC1 was the most influential.

7.7 Threats to validity

A discussion of possible threats to validity will help us to qualify the results and highlight some of the issues associated with our study.

Conclusion validity. One issue that could affect the conclusion validity is the relative small size of the sample data. Concerning data quality, the data for NLINE and NWORD have been collected using a computerised tool and are therefore reliable. The data collection for NCHANGE, NUC, NACTOR, and NREVISION involved human judgement. However, we have defined measurement rules to keep the judgment as objective as possible (see appendix A). Furthermore, the measurement rules have been tested; three subjects have independently applied the rules on two versions of a requirement document obtaining the same results. In retrospective studies there is a risk of incomplete data. In our case all files were obtained from the company’s version handling system. The project manager acknowledged that all requirements documents were handled by this system and we could not find any dangling file reference.

Construct validity. The construct validity is the degree to which the variables are accurately measured by the measurement instruments used in the study. The construct validity of the measures used for the independent and dependent variables, is ensured by the theoretical validation performed in [26].

External validity. Our threats to external validity are minimal because the study was performed in an industrial environment, therefore the materials used, and the projects themselves were real. Other factors that may affect the external validity of this study could be the project length, the experience of customers and developers in specifying requirements, the techniques used to collect and document requirements, and the company’s maturity (in requirements and software processes). The projects were developed in the same company and
used (approximately) the same processes, but everything else was different: size, personnel, and type of project. That means, we can assume that the models are valid for a relatively large class of projects.

Internal validity. The study described here is a correlational study. We have shown that all measures are significantly correlated to volatility as defined in section 4.3. However, this relationship does not imply a causal relationship. Only a controlled experiment, where the measures would be varied in a controlled manner (and the other factors would be held constant), could demonstrate causality, as we have discussed in section 4.4. One threat to internal validity is the potential inaccuracy of collected data on changes to requirements. Change requests were not available. Actual changes to requirements documents were determined by comparing all revisions of the requirements documents analysed. Rules of measurements on how to count changes were defined (see appendix A). However, we had access to and analysed all revisions of the requirements documents analysed. Version and configuration management is handled seriously at the company (see conclusion validity).

7.8 Discussions and conclusions
In this paper, we have described a correlational study on requirements volatility performed at BAE Systems Hägglunds AB, Sweden. We collected and analysed data of two historical projects in the company for the measures NUC, NACTOR, NLINE, NWORD, NREVISION and NCHANGE. Applying univariate and multivariate regression analysis, we built prediction models using data collected on a medium size software project. Only the model having NLINE as covariate was found significant. We then evaluated the model accuracy by applying it on a set of data collected on a second, slightly larger, project developed at the same company. Our prediction model receives as input the number of lines of a file describing software requirements. The model produces a number as output: the sum of the change densities in time for this requirements document. Comparing the number obtained to a threshold determined by the company, project managers can identify critical requirements and allocate resources for analysing reasons for their volatility. In this way they can minimise the risks of schedule and cost overruns.

Our hypothesis was: the size measures NACTOR, NUC, NWORD, and NLINE, are good predictors of volatility (as defined in section 4.3). The data analysis showed that the measure NLINE has a significant positive relationship to volatility. The accuracy of the measure NLINE as predictor is shown by the low value of MMRE and high value of pred (see table 6). The other measures were not significant predictors. As a result of the PC analysis and of the univariate analysis we could not combine the independent variables to construct a multivariate model. When applying model 3 on data set B, NLINE has good performance, the values of MMRE obtained are in the range of the recommended values for reliable models.

By regularly comparing the current volatility to the predicted volatility, as suggested by Costello and Liu [11], project managers can identify critical requirements and allocate resources for a closer analysis of the reasons for their volatility. In the present work, we deal with two factors of volatility: the number of changes to requirements and time. As discussed in sec-
tions 4.2 and 4.3, volatility is a quite complex concept depending on many more factors than size. For a deeper analysis of volatility it is suggested to investigate qualitative aspects such as why the changes occur, how critical the changes are, the type and phase where the changes occur. Regularly investigating many qualitative aspects on many requirements documents is, however, expensive and subjective and therefore not feasible. However, studying the impact of a change might help to "classify" changes and to identify the most critical changes. In a qualitative analysis, we would care only about important changes since all others will not affect the project much. With our prediction model we can quantify the "instability" of requirements in order to identify the critical ones. When the critical requirements are identified, we can perform a deeper analysis of the changes in order to figure out the problems with the requirement.

Although our approach currently uses use case based requirements, the defined measures are quite general and can therefore be used for all use case documents written in textual form. Furthermore, the measure NLINE can be applied to any kind of requirement written in textual form. Our approach of using counts is simple, effective, easily interpreted and completely automated.

We believe it is possible to develop prediction systems based on simple measures such as ours. The reason for our confidence is that the correlational study presented here is based on two different industrial projects. Although the exact nature of the prediction systems will vary from company to company, the underlying principle is the same. That is, developers can collect simple measures derived from requirements documents, and build effective prediction systems using techniques like linear regression analysis. Models must however be calibrated to suit different environments [30].

We plan to apply the models in larger projects in the same company and in other companies. The study described in this paper needs to be replicated in a variety of environments and systems in order to build a body of knowledge in the area.

7.9 Measurement rules

In this section, we describe the rules we adopted for measuring the requirements documents of the two analysed projects. For this study we considered files as units of requirements documents. We had full access to the repository of the projects and retrieved all existing revisions of all files analysed from the start of the projects until the point in time of the analysis.

The measure NCHANGE was obtained by counting the number of changes to a unit of requirement document. We compared two successive versions of a file using the "track changes: compare documents" tool in MSWord. Each word added or deleted was considered as one change, while each word substituted was considered as two changes (one deletion plus one addition). Furthermore, adding a picture, resizing, adding a frame, adding or deleting a detail in a picture were all considered as one change. Intuitively, adding a picture should be considered a bigger change compared to resizing it. However, if changes are made in the pictures these are usually mirrored with changes in the text.
Exceptions to these rules were the following. We did not count addition or deletion of empty space, empty lines added or deleted, page breaks or tabs without text. Automatic changes like the date in the headers and the filename in the footers were not considered changes. If a section was inserted or deleted and the successive section's numbers were changed, they were not considered as changes. The same change repeated many times in different paragraphs was considered as many changes. There were changes like a substitution of a newline where a word appears in red. We did not considered these words as changed.

7.10 References


DRAFT: Are size measures better than expert opinion? An industrial case study on requirements volatility

Annabella Loconsole, Jürgen Börstler

Abstract

Requirements volatility is an important risk factor for software projects. Software measures can help in quantifying and predicting this risk. In this paper, we present an industrial case study that investigated measures of volatility for a medium size software project. The goal of the study is to find the best measures associated with the volatility of use cases (UC) by investigating structural size measures and developers perceptions of requirements volatility. Measurement data was collected in retrospect for all use cases of the software project. In addition, we determined subjective volatility by interviewing stakeholders of the project. Our data analysis showed that structural measures perform better than subjective estimations. These results confirm our previous case study, and suggest that project managers at this company should measure their projects because of the risk to take wrong decisions based on their own and the developer’s perceptions.

Keywords. Requirements, Volatility Measures, Empirical Validation, Case Study, Use Case.
8.1 Introduction

Software project success is influenced by many factors. One of them is requirements stability \cite{20}. Volatile requirements can cause cost overruns and schedule delays and this leads to project failure. Therefore, it is important to estimate the volatility of the requirements in order to predict the risk that the project is going off track.

Software measurement can help us in providing guidance to the requirements management activities by quantifying the amount of changes and predicting the risk of frequent changes to requirements. Requirements and other entities in software engineering can be measured by observing their structural properties (like size, complexity, functionalities), or through subjective evaluation. An interesting question that, to our knowledge has not been investigated in depth to date, is whether structural measures can perform as well as or better than experts in predicting quality attributes \cite{3}. Subjective evaluations and expert opinion are not commonly used in software engineering and one reason of this can be the difficulty of collecting reliable data \cite{20} (see wohlin and aaa ESE 2003 for related work on subjective evaluation).

In this paper, we describe an industrial case study that investigated the requirements volatility of a software that runs on automotive systems. Our goal is to find the best measure of requirements volatility. This is done by investigating requirements size measures and perceptions of volatility and correlating them to objective measures of volatility (see table 1). In the present study, we analysed data in retrospect for 21 UCs for a medium size software project and interviewed the stakeholders of the project (professional developers) about their perception of requirements volatility.

This work is a replication of a case study described in \cite{14}, both were performed at BAE Systems Haggunds AB, Sweden. The reason for performing a replication is to confirm the previous results. Many studies are isolated and it is hard to understand how widely applicable are the results of these isolated studies. Through a replication we can increase the applicability of the results of our previous case study. The study we present in this paper is not a strict replication, because we vary slightly the research hypothesis and the manner in which the experiment is done \cite{1}.

The remaining part of the paper proceeds as follows: in section 2 we briefly summarise our previous empirical studies, which investigated the relationship between measures of size of requirements and number of changes to requirements. Section 3 describes the goals, hypotheses, and data collected in the present correlational study. The data analysis is described in section 4. The threats to validity are presented in section 5. A comparison of the two studies and a discussion is described in section 6. Finally, conclusions and future works are presented in section 7.

1Sometimes, the word stability is used instead of volatility. For instance, a definition of “degree of stability” of requirements is presented in \cite{10}. Both terms are used conjunctly in \cite{8, 17}. In our opinion, the words are antonyms.

--general about volatility and the importance of measuring it
--something about measurement though perception and expert opinion?
--theoretical and empirical validation has to be done for the dependent and independent attributes. dangers of using not validated measures
--importance of replication of empirical studies.
--Something about agile approaches and volatility (in the limitations or validity)?

8.2 Goals of the study

The goal is to see if the results of the Apsec study still hold in another project (are perception good estimators of volatility? Are requirements measures good estimators of volatility?) (for the second hypothesis the measures I will use are slightly different from previous study: num of actors, flows, words, and lines. In the previous study was used num of use cases and not num of flows)

8.3 Theoretical validation

Check that both size of UC and size of change satisfy the 3 properties of size (nonnegativity, null value, additivity).

8.4 Empirical study description

We followed the process suggested by Wohlin et al. []. All materials of the study will be available online 5 to allow for possible replications.

8.5 Definition

Following the GQM template [] for the definition of the goals, we obtain the following definition for our case study: Analyse the use cases of the information system of an automotive system. For the purpose of 1. investigating the predictive accuracy of perceptions and 2. to investigate the predictive accuracy of requirements measures With respect to volatility of requirements. From the point of view of the academy. In the context of an industrial environment.

The second goal of the study is to validate a set of requirements volatility measures empirically..... In our case study we can only test for correlation (and not causality) due to low control compared to formal experiments. The first goal.....

5. www.cs.umu.se/~bella/......................
8.6 Planning

Context selection

The case study was performed in retrospect at Land Systems Hägglunds (BAE Systems) Sweden. The company produces automotive systems with embedded software and has ISO9001 and ISO14001 certifications. The project chosen for the study (host project), builds the information system that runs on a tank. About twenty people have worked on the software development project two project managers, and eight developers. The system comprises of 24-26 use cases. According to the project terminology a use case description contains: overview, revision history, references, description, state-diagram, normal flow, alternative flows, special requirements, start conditions, end conditions, and extension points.. The description of the actors was contained in another file at a higher abstraction level.

The project started during summer 1999 and ended in 2003 (?). The time delay between the project and the case study was about two years.

The context selection of the case study can briefly be characterized according to four different dimensions: 1) online because it is performed in an industrial software development environment; 2) professional i.e. non-student environment; 3) real because it addresses a real problem; 4) specific since it is focused on volatility measures.

Selection of subjects and objects

In order to determine the perceived volatility, we contacted 42 stakeholders at the company. However, not all of them were involved with software development. Around 20 were sw developers, other people had different roles like market responsible, project managers, quality assurance responsible, system engineers etc. No users were contacted (because I was not allowed to do that).

The objects used in the study are: email form, Microsoft excel and word for comparison, minitab for the analysis.

| TABLE 18: Internal attributes and measures of volatility |
|-------------------------------------------|------------------|
| | Size of UC | Size of change to UC |
| | number of lines | number of changes |
| | number of words | number of revisions |
| | number of actors | |
| | number of associations or flows | |

Variables selection

The independent variables were the internal attribute size of UC, and perceived volatility. The dependent variable is the number of changes to use cases. The measures are described in table 18. The measures are quite intuitive, except for number of revisions. Revisions
are done to any files when changes are performed, but also to validate the UCs. A revision can include several changes or no changes at all. Usually, a large amount of revisions corresponds to a large amount of changes.

Perceived volatility was measured by subject ratings, which were collected manually by an e-mailed form (see appendix A). In order to determine the perceived volatility, we contacted 44 stakeholders at the company. Among them were xx project managers (of which one was also a developer), xx developers and xx internal users of the software system. External customers were not interviewed.

One possible measure of volatility could also be “number of associations between use cases”, which is associated to the internal attribute “complexity of use case model” (note: this is also a measure of size!). The measurement rules, applied when collecting data, are described in the appendix. Although we are aware of the difficulty of determining the exact scale type [], our measures seem to belong to the ratio scale.

**Instrumentation**
The objects of the study were 22 UCs of the project at the company.

**TABLE 19:** Data collected on UCs

<table>
<thead>
<tr>
<th>UC</th>
<th>Num of revisions</th>
<th>Total Num of changes</th>
<th>Num of flows</th>
<th>Num of actors</th>
<th>Num of lines</th>
<th>Num of words</th>
<th>Subjective volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uc1</td>
<td>4</td>
<td>1526</td>
<td>4</td>
<td>6</td>
<td>136</td>
<td>1145</td>
<td></td>
</tr>
<tr>
<td>Uc2</td>
<td>2</td>
<td>1223</td>
<td>4</td>
<td>4</td>
<td>200</td>
<td>1428</td>
<td></td>
</tr>
<tr>
<td>Uc3</td>
<td>2</td>
<td>587</td>
<td>3</td>
<td>4</td>
<td>146</td>
<td>813</td>
<td></td>
</tr>
<tr>
<td>Uc4</td>
<td>11</td>
<td>2587</td>
<td>2</td>
<td>4</td>
<td>409</td>
<td>2451</td>
<td></td>
</tr>
<tr>
<td>Uc5</td>
<td>5</td>
<td>3715</td>
<td>0</td>
<td>4</td>
<td>248</td>
<td>1168</td>
<td></td>
</tr>
<tr>
<td>Uc6</td>
<td>5</td>
<td>1207</td>
<td>1</td>
<td>4</td>
<td>181</td>
<td>1412</td>
<td></td>
</tr>
<tr>
<td>Uc7</td>
<td>4</td>
<td>383</td>
<td>3</td>
<td>4</td>
<td>110</td>
<td>799</td>
<td></td>
</tr>
<tr>
<td>Uc8</td>
<td>2</td>
<td>1038</td>
<td></td>
<td>2</td>
<td>168</td>
<td>980</td>
<td></td>
</tr>
<tr>
<td>Uc9</td>
<td>3</td>
<td>280</td>
<td>2</td>
<td>4</td>
<td>110</td>
<td>647</td>
<td></td>
</tr>
<tr>
<td>Uc10</td>
<td>2</td>
<td>115</td>
<td>0</td>
<td>1</td>
<td>38</td>
<td>187</td>
<td></td>
</tr>
<tr>
<td>Uc11</td>
<td>2</td>
<td>375</td>
<td>1</td>
<td>4</td>
<td>80</td>
<td>390</td>
<td></td>
</tr>
<tr>
<td>Uc12</td>
<td>3</td>
<td>227</td>
<td>0</td>
<td>4</td>
<td>85</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>Uc13</td>
<td>2</td>
<td>1540</td>
<td>6</td>
<td>2</td>
<td>240</td>
<td>1501</td>
<td></td>
</tr>
<tr>
<td>Uc14</td>
<td>2</td>
<td>239</td>
<td>4</td>
<td>4</td>
<td>57</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>Uc15</td>
<td>3</td>
<td>407</td>
<td>4</td>
<td>4</td>
<td>131</td>
<td>877</td>
<td></td>
</tr>
<tr>
<td>Uc16</td>
<td>2</td>
<td>213</td>
<td>2</td>
<td>2</td>
<td>50</td>
<td>288</td>
<td></td>
</tr>
<tr>
<td>Uc17</td>
<td>2</td>
<td>490</td>
<td>1</td>
<td>2</td>
<td>101</td>
<td>547</td>
<td></td>
</tr>
<tr>
<td>Uc18</td>
<td>2</td>
<td>362</td>
<td>2</td>
<td>2</td>
<td>86</td>
<td>361</td>
<td></td>
</tr>
<tr>
<td>Uc19</td>
<td>2</td>
<td>456</td>
<td>3</td>
<td>2</td>
<td>101</td>
<td>565</td>
<td></td>
</tr>
<tr>
<td>Uc20</td>
<td>3</td>
<td>377</td>
<td>3</td>
<td>2</td>
<td>108</td>
<td>481</td>
<td></td>
</tr>
</tbody>
</table>
Other documentation was used to gather data about the objects, in particular project plans, iteration plans, and test plans. The guidelines for the subjects were described in an email sent to all participants (see the appendix or maybe an internet web address). The measurement instruments used were Microsoft Excel forms.

The external attribute was measured by collecting the ratings of volatility made by the subjects. Measurement was performed through manual data collection (email form).

Hypotheses formulation.
Our null hypotheses were the following:

- There is no significant correlation between the objective and subjective rating of volatility of the UCs (There is significant correlation between the measures of size of change to UC and the rating of volatility of the UCs made by the subjects.)
- Second hypothesis: the measures number of actors, number of flows, number of lines, and number of words are good predictors of requirements volatility.

Experiment design

To test the first hypothesis we chose a within-subject design (i.e. all subjects filled in the same form). For the second hypothesis we chose correlation [ ].

8.7 Operation

Preparation
The preparation of the subjects was made by explaining the definition of requirements volatility, describing the forms (shown in appendix) and showing an example of how the form should be filled in (see the appendix). Even if the example shown is for three UCs, the set of 22 use cases plus one file containing 5-6 non functional requirements is quite small and the example can easily be extended to the whole set. The subjects were not aware of the hypotheses. We handed in the material with 24 UCs of the project, plus one file with NFR providing only their descriptive names. Each subject worked alone (infact the subjects did not know who was participating to the study) and could use unlimited time. They rated the relative volatility defined as trends in changes to UCs in the RUP phases inception, elaboration, construction, and transition. The rate was on a scale of three linguistic labels (low, medium, and

### TABLE 19: Data collected on UCs

<table>
<thead>
<tr>
<th>UC</th>
<th>Num of revisions</th>
<th>Total Num of changes</th>
<th>Num of flows</th>
<th>Num of actors</th>
<th>Num of lines</th>
<th>Num of words</th>
<th>Subjective volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>UC21</td>
<td>2</td>
<td>172</td>
<td>0</td>
<td>1</td>
<td>32</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>UC22</td>
<td>2</td>
<td>170</td>
<td>0</td>
<td>1</td>
<td>31</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>NFRs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
high) plus a “do not know” for those who have partially been involved in the project or for other reasons. We asked also a total volatility for each use case over the phases, and a total volatility for each phase for all the use case.

Creating a form with four linguistic labels allowed us to fit the form on a single page that was easy and quick to fill in, in order to encourage the subjects to participate in the study. We asked the subjects to not read the documentation of the UCs in order to avoid letting the objective volatility affect the perceived volatility. This documentation contains some information (like the number of revisions, the description and the date of changes to the UC) which may hint the subjects about the volatility of the UC.

The study was divided into two phases: the first part consisted of manual collection of data for the measures in table 3. We collected data starting from the first available revision of the UCMs which were dated July 2001 (three months after the beginning of the project). In the second phase we distributed the forms shown in table 5.

**Execution and data validation**

The study did not affect the development project because it was done in retrospect. We collected the data by reading historical project documentation. We contacted 42 stakeholders at the company for the subjective data, and received answers from 7 of them.

### 8.8 Analysis and interpretation

To verify the two hypotheses we checked the distributions of data for normality. As the data distribution were not normal, we used non-parametric statistics and applied the Spearman correlation coefficient. We chose a level of significance $\alpha = 0.05$, i.e. the level of confidence is 95%. For a sample of size 22, the Spearman cutoff for accepting $H_0$ is xxx. However, as the formality in our case study is low compared to formal experiments, we consider the Spearman cutoff only a reference point to judge the significance of our correlations.

**TABLE 20:** Spearman correlation coefficient obtained analysing the two hypotheses

<table>
<thead>
<tr>
<th></th>
<th>Total number of changes</th>
<th>Number of revisions</th>
<th>Subjective volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of lines</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of words</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of actors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of flows</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subjective volatility</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Testing the first hypothesis.**

subjective versus objective volatility (can be used to check how bad are subjective predictions compared to my prediction models)
Since the rating of volatility made by the subjects yielded ordinal data we applied a simple transformation (weighted average) to obtain ranked data. For the current analysis, we averaged the answers for the phases to one value for each UCM. Each of the measures was correlated separately with the transformed rating of volatility made by the subjects (last row and last column in table 20). Because all coefficients are below/above the cutoff, we can conclude that there is no significant correlation between our measures and the subjective rating of volatility.

**Testing the second hypothesis.**

Possible analysis: spearman, pearson, PCA, univariate regression, multivariate regression, ..... with the measures number of actors, number of flows, number of lines, and number of words (without num of use cases that was used in the previous case study).

The number of change to UC was correlated separately with the size of UCM measures. As we can observe in table 20, the values in bold show high/low correlation. The measure total number of changes is high/low correlated with the measures of size of UC. Concluding, there is a high/low correlation between the size of UC and the total number of changes to it. For the moment, we do not claim any causality (i.e., that larger use cases cause a higher number of changes). Further studies are needed to check the causality and to identify guidelines for optimal size of UCMs from a volatility perspective.

### 8.9 Validity evaluation

#### 8.10 Replication: comparison of the two studies

Many studies are isolated and it is hard to understand how widely applicable are the results of these isolated studies. Through a replication we can increase the applicability of the results of our previous case study. However, even when replications are run it is sometimes hard to understand the commonalities and the differences so that we can draw general conclusions. Therefore, in this section we will describe the differences among the two studies and draw some conclusions.

Replications of experiments can be classified in 3 major groups: replications that do not vary any research hypothesis, the ones that vary research hypothesis, and the replication that extend the theory. Our case study is an internal replication (because it is performed in the same company), which falls in the second group. However, it is not a strict replication, because we vary slightly the research hypothesis and the manner in which the experiment is done. In performing the internal replication, we have done some changes (to

---

6. We counted the number of occurrences of each answer and multiplied the number obtained by 0 (low), 0.5 (medium), or 1 (high). Finally we summed the result and divided by 6 (the max number of subjects).
the hypotheses, variables, and instrumentations in order to compensate for the threats. The problem is that the attempt to compensate for threats to validity may lead to changes such that the replication becomes less strict and therefore it is difficult to generalise.

A summary of the differences between the two studies can be seen in table 4. Differences between the studies concerning the state and response variables have been described in section \ref{variables}.

The stakeholders contacted in both studies were employees at the company with different roles (from software developers to project managers to hardware developers).

The material we distributed to the subjects is similar. In case study A we delivered a form where we described the definitions of volatility and relative volatility. In case study B we added the definitions of requirement and change because...... In case study B the form to be filled in was divided by the RUP phases instead in case study A it was divided by the four main waterfall phases (requirement, design, coding and maintenance). Furthermore, in case study B the subjects could choose among low, medium, high, and don’t know, while in the previous study there were only 3 choices (low, medium, high). The material was given in an email form in both studies.

The measurement rules for counting changes are different. (%can we really say that it is a replication?).

In case study A we classified changes in minor, moderate, and major. Furthermore, zero changes were associated to the first revision of each requirements document. In case study B we counted only the changed words. To the first revision, it was associated an amount of changes equal to the number of words of the whole requirement document. These differences affected the choice of the response variable in case study B. Differences in the context of the study and in the datasets are described in \cite{loconsole07}.

From an analysis of the data collected for the state and response variables common to the two studies, we obtain the following results:

1. Number of lines and words show high correlation to number of changes in both studies.
2. Number of actors shows correlation in case study A but not in case study B.
3. Subjective volatility do not show any correlation to number of changes in both studies.

However, we cannot consider these results as definitive, further empirical studies are needed in order to confirm or falsify these results.

inducement (cinema tickets). Project size, type of project, number of participants,

<table>
<thead>
<tr>
<th>TABLE 21: differences among the studies</th>
</tr>
</thead>
<tbody>
<tr>
<td>latest case study (described here)</td>
</tr>
<tr>
<td>use cases</td>
</tr>
</tbody>
</table>
8.11 Conclusions

In this paper, we have described a retrospective case study on requirements volatility performed at BAE Systems Haglund’s AB, Sweden. We collected data for four structural measures of size and measures of change to UCs, associated to the external process attribute requirements volatility for a medium size software project. We furthermore interviewed project stakeholders about their perception of requirements volatility.

Summarizing the results, we found that:
the measures of length of UCs (number of lines and words) are highly correlated with the measure total number of changes to UCs while number of actors and number of scenario did not show a significant correlation. In case study A, we found that all the requirements size measures were significantly correlated to total number of changes. Therefore, the length measures seem to be best estimators of total number of changes. The correlation among them is linear \cite{loconsole07}. This result supports the intuitive notion that larger UCs are more volatile than smaller ones and should encourage developers to modularize UCs.

This serves to re-emphasize some fundamental software engineering principles for the need of modularity and cohesion in order to manage complexity and localize change. However, further studies are needed to identify guidelines for optimal size of UCMs from a volatility perspective.

All the structural size measures did not show any significant correlation to volatility.
However, although not significant, the length measures are positively correlated to volatility compared to the subjective estimations and the other measures. This result complies with our correlational study \cite{loconsole07}02 where we found that number of lines is a good predictor of volatility on both projects, as defined in section \ref{variables}.

The subjective volatility did not show any significant correlation to total number of changes (in both studies) and volatility (in case study B). Once again, perception did not match our measures.

This implies that decision makers may take a high risk when basing decisions solely on subjective requirements volatility. Therefore, we suggest that project managers at this company measure their projects in order to minimize this risk.

Although our approach currently uses use case based requirements, the defined measures are quite general and can therefore be used for all use case documents written in textual form. Furthermore, the measure NLINE can be applied to any kind of requirement written in textual form. Our approach of using counts is simple, effective, easily interpreted, and completely automated.

Our results are based on two industrial projects and therefore have local validity. However, the projects analysed were quite different from each other (the systems developed, the developers, the size), therefore, we are quite confident that the results can be generalised. However, the study described in this paper needs to be replicated in a variety of environments and systems in order to build a body of knowledge in the area.

8.12 Future work
investigate relative volatility, trends, reasons of change, different abstraction levels, change requests and trouble reports.

8.13 References
[1] basili shull lanubile ieee tse 25(4)
[2] Briand, el emam morasca- theoretical empirical validation
[3] briand morasca basili- property based...
[4] kitchenham, fenton, pfleeger- towards a framework...
[5] loconsole apsec
[6] loconsole JSS
8.14 Appendix

8.15 Measurement rules

In this section, we describe the definitions and rules we adopted for measuring the UCs and the NFRs. For this study we considered UCs as units of requirements. Each UC was described in a single file (see figure for the template). The NFRs were described in a single file. We had full access to the repository of the project and retrieved all existing revisions of all UCs from the start of the project until the current date.

Measure of size of a UC and of NFRs

We measured the size in different ways: number of lines and words in the file. These two measures were used for both functional and non-functional requirements. Number of flows and number of actors instead served only to measure use cases. We used the MS Word count tool to count lines and words.

The number of use cases and actors will be measured at a higher abstraction level. The information of the actors and use cases is not contained in the same file as the use cases description.

Measure of number of changes.

This measure was counted in two ways: by counting the total, and the number of revisions which correspond to different versions of a file with an identifier.

The size of change is usually dependent on the effort spent for the change or the number of artifacts impacted by the change. Unfortunately, this information was not available in the current project. Therefore, the size of change was estimated by the authors. In counting the changes we did not have a database of changes to UCs nor to NFRs. We counted the changes manually using the following counting procedures and rules. We compared two successive versions of a UC file using the “track changes: compare documents” tool in MS Word. A sentence highlighted in red or several words changed in one paragraph were considered as one change. Exceptions to this rule were the following.

- We did not count changes in the date in the header.
- We did not count addition or deletion of white space, empty lines added or deleted, page breaks or tab without text.
- If a paragraph was inserted or deleted and the successive paragraph numbers were changed, they were not considered as changes.

Some problems were encountered while measuring. For instance, how to consider the same change repeated many times in different paragraphs. In both cases we decided to consider them as many changes. Another problem was caused by the fact that the project was divided in two different projects.
Instructions for the subjects.

DEFINITIONS

- Requirement = a functionality (usually a use case) or a quality of the software system
- Change = any modification to the requirements (even small modifications like one word changed)
- Volatility = the amount of changes to a requirement in time
- Relative volatility = the volatility of a requirement compared to other requirements.

For each requirement and RUP (rational unified process) phase, please indicate its relative volatility in the CV9030 Finnish and Swiss project by placing a letter (L, M, H, D) in each box.

L = low
M = medium
H = high
D = do not know

More concretely, you should compare the amount of changes to each requirement to the amount of changes in average. In case there were differences between requirements for the Swiss and Finnish project you can indicate it by placing Ch or Fi.

EXAMPLE:

TABLE 22: form distributed to the subjects

<table>
<thead>
<tr>
<th>What is the volatility of the following requirements?</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
<th>In general for all phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>requirement1</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>requirement2</td>
<td>M</td>
<td>M</td>
<td>H(FI m(CH))</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>requirement3</td>
<td>L</td>
<td>L</td>
<td>M</td>
<td>M</td>
<td>L</td>
</tr>
<tr>
<td>In general for all requirements</td>
<td>D</td>
<td>M</td>
<td>L</td>
<td>H</td>
<td>D</td>
</tr>
</tbody>
</table>

where:

- inception = correspond to planning and initial requirements specification phase
- elaboration = more detailed requirements specification and initial design
- construction = detailed design of the system, coding and testing phases
- transition = deployment, transferring the system to the customer.
Thus, the meaning of the example table given above is the following: in the inception phase, the amount of changes to requirement 1 is high in relation to requirement 2 and it is very high in relation to requirement 3 for both the Swiss and Finish projects. In the elaboration phase, the amount of changes to requirements 1 and 2 is higher in relation to requirement 3. The volatility of requirement 2 in the construction phase was higher for the Finish than the Swiss project. And so on.

Here is the form I would like you to fill in (here there are the filled forms):

TABLE 23:

<table>
<thead>
<tr>
<th>What was your role in the project?</th>
<th>In which phase did you start to work on the project?</th>
<th>Did you work with software requirements?</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA: Part of the VIS Team Defining the use cases Design Implementation</td>
<td>Elaboration phase</td>
<td>Yes</td>
</tr>
<tr>
<td>CL: Project manager electrical system</td>
<td>Elaboration</td>
<td>My fill in below are based on the work with power point presentation.</td>
</tr>
<tr>
<td>JB: Project manager, developer</td>
<td>Inception</td>
<td>Some, but mostly HW for DIS</td>
</tr>
<tr>
<td>SE: Design manager</td>
<td>In the Construction phase</td>
<td>No, focusing on electrical/electronic hardware</td>
</tr>
<tr>
<td>TS: Technical coordinator &amp; design manager for the electronic and software part of the system</td>
<td>From the early beginning it means inception</td>
<td>I worked with requirements that led to software based solutions. I was also involved with software requirements.</td>
</tr>
<tr>
<td>BY: Hardware developer</td>
<td>From the beginning</td>
<td>Very little</td>
</tr>
<tr>
<td>OT: Project manager and requirements, quality, documentation and more</td>
<td>Second half of 1999</td>
<td>yes</td>
</tr>
<tr>
<td>DM: Testing of software and electronics</td>
<td>Late construction phase</td>
<td>Yes, when updating test specifications, trouble shooting etc</td>
</tr>
</tbody>
</table>


TABLE 24:

<table>
<thead>
<tr>
<th>What is the volatility of the following requirements?</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
<th>In general for all phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE 24:</td>
<td>What is the volatility of the following requirements?</td>
<td>Inception</td>
<td>Elaboration</td>
<td>Construction</td>
<td>Transition</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Välja profil</td>
<td>D,L,D,D,M,D,N, A</td>
<td>L,I,D,D,I,L,NA</td>
<td>L,I,D,D,I,L,L</td>
<td>L,I,D,D,I,L,D,--</td>
<td>L,I,D,D,I,L,D,--</td>
</tr>
<tr>
<td>Välja videokälla</td>
<td>D,L,D,M,D,D, NA</td>
<td>L,I,D,M,L,M,NA</td>
<td>L,I,D,M,L,L,I</td>
<td>L,D,I,L,L,--</td>
<td>L,D,M,I,L,D,--</td>
</tr>
<tr>
<td>Se på video</td>
<td>D,L,D,M,D,D, NA</td>
<td>L,I,D,M,L,M,NA</td>
<td>L,I,D,M,L,L,I</td>
<td>L,D,I,L,L,--</td>
<td>L,D,M,I,L,D,--</td>
</tr>
<tr>
<td>Hämna DTCer</td>
<td>D,L,D,I,D,M,N, A</td>
<td>H,I,D,M,M,D,NA</td>
<td>M,M,D,M,M,L</td>
<td>L,I,D,L,L,D,--</td>
<td>M,L,D,M,D,--</td>
</tr>
<tr>
<td>Radera HSV-interna loggar</td>
<td>D,L,D,L,D,N, A</td>
<td>H,I,D,M,L,D,NA</td>
<td>M,I,D,H,M,D,M</td>
<td>L,I,D,L,M,D,L</td>
<td>M,I,D,L,D,--</td>
</tr>
</tbody>
</table>
In the end of this file you will find more details on the requirements i.e. the filename, use cases name in Swedish and in English.

When filled in, please send this form to bella@cs.umu.se

Thank You for your cooperation.

**TABLE 24:**

<table>
<thead>
<tr>
<th>what is the volatility of the following requirements?</th>
<th>inception</th>
<th>elaboration</th>
<th>construction</th>
<th>transition</th>
<th>In general for all phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplementary specification information system</td>
<td>D,M,D,D,L,M,--</td>
<td>H,H,D,D,M,M,--</td>
<td>H,L,D,D,H,M,--</td>
<td>L,L,D,D,M,M,--</td>
<td>M,L,D,M,M,--</td>
</tr>
</tbody>
</table>

In the end of this file you will find more details on the requirements i.e. the filename, use cases name in Swedish and in English.

When filled in, please send this form to bella@cs.umu.se

Thank You for your cooperation.

**TABLE 25:** non rank (pearson correlation)

<table>
<thead>
<tr>
<th></th>
<th>nfloows</th>
<th>actors</th>
<th>lines</th>
<th>words</th>
<th>subjective</th>
<th>revis</th>
</tr>
</thead>
<tbody>
<tr>
<td>nchange</td>
<td>-0.038</td>
<td>0.330</td>
<td>0.831</td>
<td>0.764</td>
<td>-0.102</td>
<td>0.663</td>
</tr>
<tr>
<td>change density</td>
<td>-0.522</td>
<td>-0.247</td>
<td>0.080</td>
<td>-0.029</td>
<td>-0.315</td>
<td>0.126</td>
</tr>
<tr>
<td>volatility</td>
<td>-0.529</td>
<td>-0.185</td>
<td>0.164</td>
<td>0.046</td>
<td>-0.338</td>
<td>0.222</td>
</tr>
<tr>
<td>revisions?</td>
<td>-0.178</td>
<td>0.367</td>
<td>0.787</td>
<td>0.749</td>
<td>-0.243</td>
<td>1</td>
</tr>
</tbody>
</table>

**TABLE 26:** rank (spearman correlation) 0.425

<table>
<thead>
<tr>
<th></th>
<th>nfloows</th>
<th>actors</th>
<th>lines</th>
<th>words</th>
<th>subjective</th>
<th>revis</th>
</tr>
</thead>
<tbody>
<tr>
<td>nchange</td>
<td>0.276</td>
<td>0.338</td>
<td>0.941</td>
<td>0.944</td>
<td>0.208</td>
<td>0.443</td>
</tr>
<tr>
<td>change density</td>
<td>-0.366</td>
<td>-0.321</td>
<td>0.179</td>
<td>0.172</td>
<td>-0.097</td>
<td>0.019</td>
</tr>
<tr>
<td>volatility</td>
<td>-0.425</td>
<td>-0.282</td>
<td>0.192</td>
<td>0.181</td>
<td>-0.181</td>
<td>0.128</td>
</tr>
<tr>
<td>revisions?</td>
<td>-0.225</td>
<td>0.537</td>
<td>0.526</td>
<td>0.496</td>
<td>-0.238</td>
<td>1</td>
</tr>
</tbody>
</table>