Validation Of Requirements Management Measures

Master Thesis

Mohammad Abubakr Saeed

ens03asd@cs.umu.se

October 18, 2004

Department of Computing Science
Umeå University
Umeå
Sweden

Abstract

Requirement management is a vital and important activity of software development. The aim of requirements management within any development organization is to provide consistency, predictability, and control over requirements. Measurements provide insight into several aspects of requirements development and management activities. There are numerous requirement management measures that claim this objective, however few of them have been validated. Software measures validation is used to ensure that the measure is really measuring the object it is purports to measure. The objective of this dissertation is to validate eight requirement management measures defined in [Loconsole, 01]. This is done in two ways: the first method is to conduct a survey to know opinions and experiences of software developers related to requirement management measures. Some software developers across different countries were interviewed using a questionnaire method by mean of phone, email and in person. The second way to validate the measures is by applying the theoretical validation procedures.

We have not been so successful to collect massive data and conclude better results from survey, however we can still show some conclusions. From the views of software developers it is concluded that measures are in use to some extent but still there is much need to practice requirement management measures to establish the correctness of this phase. For the second way to validate the eight requirements measures it was difficult to know which view of validity is widely accepted and have the consensus of the software community.

Acknowledgements

I would like to thank my supervisor Annabella Loconsole for her help, support, valuable suggestions and timely advices, which played an important role in the fulfillment of this thesis. I would also take an opportunity to thank all the nice people that I have interviewed.
Least and not less I would like to thanks R.S.Rehman, T.Blomqust and M.Jamal for their help which I probably never will excel. Without them this paper would never exist.
Thanks to my family for financial and moral support when things looked dark.

TABLE OF CONTENTS

Table of Figures and Tables

1 Introduction

1.1 Background

Requirements management involves establishing and maintaining agreement between customer and developer on both technical and non-technical requirements. This agreement forms the basis for estimating, planning, performing, and tracking project activities throughout the project and for maintaining and enhancing developed software.
The main objective of requirement management is to ensure that customer and developer have a common understanding on requirements ensure that requirements must be of good quality, and requirement changes must be controlled. To perform the requirement management phase, precise information and guidelines are required for managers to make their work easier, like in decision-making, controlling, and allocation of resources for different software activities that take place during software development. Requirements management is a key phase, which has connection with quality management. In this activity we ascertain what the customer wants in terms of quality and we evaluate the solution in terms of whether it meets the quality requirements efficiently. And quality can be defined as "conformance to requirements". Effective requirements management includes requirements measures because they provide insight into the effort and time the requirement management or individual tasks consume and the product's quality. Since requirements are an essential project component, you should measure several aspects of requirements development and management activities to help in judging product size and project status. There are also measures to get control over requirement management activity, for better performance.
Measurement is a key factor in software development, basically measurement is used for these reasons:

1. It helps an organization to understand what is happening during development and maintenance.
2. It allows control over what is going on in project development.
3. It encourages improvement of processes and products.

Measurements are the scale, through which maturity of object is measured, as Fred S. Roberts said:

"A major difference between a "well developed" science such as physics and some of the less "well-developed" sciences such as psychology or sociology is the degree to which things are measured" [Roberts, 79].

In the area of software engineering, measurement is one of the activities, where researchers have been active since more than thirty years. Researchers are concerned about software quality and need to quantify it and they require having precise, predictable, and repeatable control over the software development process. For that purpose during the past, more than one thousand software measures were proposed by researchers and practitioners, and till today more than 5000 papers about software

measurement are published. Among these software metrics it is difficult to know which one is representing accurately those attributes that it purports to quantify and which measures are widely accepted. That's why it is necessary that the measures should be valid which are using for developing and delivering reliable, usable software within budget. Validity of measures ensures that measures are representing accurately those attributes they purport to quantify [Zuse, 95].

1.2 Goals and Methodologies

A set of 38 requirement management measures was defined in [Loconsole, 01]. Ten of these measures have been theoretically validated in [LoconsoleB, 03]. The primary Goal of this thesis is to validate eight requirement management measures from the set defined in [Loconsole, 01]. This is done in two ways:

1. By conducting survey.
2. By theoretical validation.

Different software developers from Sweden, India, and Pakistan were interviewed using different means i.e. phone, email, etc. The purpose of this strategy was to interview people from different geographical, educational, ethnic backgrounds and to analyze different school of thoughts and different problem solving techniques. To interview them, a questionnaire method was adopted.
For the theoretical validation of eight requirement management measures, we have been focused on a set of "38 requirement management measures" defined in [Loconsole, 01]. These measures were obtained by studying the "Requirement management" Key Process Area (KPA) of the SW-CMM [SEI, 93]. Ten of them have already been validated [LoconsoleB, 03]. In this dissertation eight of the remaining measures will be validated. This is done by using a theoretical validation framework, [KitchenhamPF, 95], and the key stages of formal measurement [FentonP, 98].

1.3 Results

By conducting the survey, we investigate the experience of software developers in practicing requirement management measures and how important is for them to measure requirements.
In applying the first method, we have not been so successful to collect massive data and conclude better result from survey. Foremost difficulty encountered is was to gather data of developers both from Europe and Asia.
However we can still show some conclusions. From the views of software developers it is concluded that measures are in use to some extent but developers also use their own experiences to performing requirement management activity rather than to follow some measures. There is still need to practice requirement management measures to establish the correctness of this phase.
In applying the second method to validate the eight requirements measures we encountered some difficulties, like it was difficult to know which view of validity is widely accepted and have the consensus of the software community. Secondly, verifying

representation condition was difficult because representation condition needs to be verified in the context of an empirical study having real requirements specification but our situation was not same. Therefore in this thesis eight of the measures have been validated. Remaining twenty measures needs to be validated in future work.

1.5 Outline of thesis

This dissertation is divided in to the following sections:

Section1: presents the introduction to the thesis. It also describes the goals of the dissertation and methodologies that have been used to accomplish these goals.

Section2: describes the concepts of software measurement, summarizes types of measurement validations, focusing on the theoretical validation procedures.

Section3: Presents the requirement management measures. And also mentions the views and experience of software developers about presented requirement management measures.

Section4: Describe a theoretical validation approach to validate requirement management measures and validate eight of them.

Section5: Contains concluding remarks and summary.

2 Software measurement

2.1 What is measurement?

Measurement is the process by which numbers or symbols are assigned to attributes of entities in the world according to a set of clearly defined rules. Once this association has been made, then it seems possible to compare the physical entities by comparing the associated numbers. This comparison leads to certain measurements. For instance, age is common attribute of the entity person. The measure can be number of years or number of months or something else [FentonP, 98].

"When you can measure what you are speaking about, and express it into numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science" [Kelvin, 89].

The important thing is that everyone means the same thing when referring to a certain measurement, therefore measurements must be standardized to mean the same thing to everyone.
Software development includes different phases like planning, designing, analyzing, coding, testing, budget scheduling, etc. Software measurement makes it possible to assess the software development process and incase some thing is wrong, it allows making adjustments to the process. The costs spent in development and maintenance of software is high therefore through software measurement we can estimate actual cost and predict future costs of development activities. It provides reliable information and insight needed by manager to make better decisions.

2.2 Objectives of Software measurement

Software measurement runs through all phases of software development. Measurement is needed for assessing the status of projects, products, processes and resources.
According to Fenton and Pfleeger [FentonP, 98] in software engineering there are three classes of entities to be measured:

Processes: set of activities, which are performed during software development, for example maintenance.
Products: any artifacts, deliverables or documents that result from process activity, for example source code.
Resources: entities essential for process activity, for example people, tool.

Projects must be controlled, not just run them. According to Fenton and Pfleeger:

"*You can not control what you can not measure* and y*ou can neither predict nor control what you can not measure"*. [FentonP, 98]

Software measurements support software development and maintenance activities and provide strong motivation for any organization to initiate or expand its analysis of data and application of results. There are three key reasons for an organization to measure its software engineering processes, product and resources.

1. Understanding
2. Control
3. Improvements

Any of these reasons should be enough to motivate an organization to implement a measurement program.
Knowing what an organization does and how it operates is a fundamental requirement for any attempt to plan, manage, or improve. Measurement provides the mechanism available for quantifying a set of characteristics about a specific environment or for software in general. Increased understanding leads to better management of software projects and improvements in the software engineering process.
The second reason for software measurement is to control our product and process. Understanding the software engineering process leads to better management decision-making. By applying measurement program, one of the most important elements of project management is achieved, which is project estimation for cost, schedule, staffing requirements, resource requirements, and risk. Successful, effective management requires visibility into the progress and general status of the ongoing project, so that timely and informed adjustments can be made to schedules.
The primary focus of any software engineering organization is to produce a high-quality product within schedule and budget. But, a constant objective must be continual improvement in the quality of its products and services. *Product* improvement can be achieved by improving the processes used to develop the product. *Process* improvement, which requires introducing changes to the process, may be accomplished by modifying management or technical processes or by adopting new technologies. In any case, software measurement is a key part of any process improvement program; knowing the quality of the product developed using both the initial and the changed process is necessary to confirm that improvement has occurred. There are several popular paradigms for software process improvement. For example, the Capability Maturity Model (CMM) [SEI, 93].

2.3 Basic definition of software measurement

The structural model of software measurement [KitchenhamPF, 95] allows describing the (real world) objects involved in measurement and their relationships. In this model, various elements contribute to measurement and relationships among them. These elements, which constitute the model, are stated below.

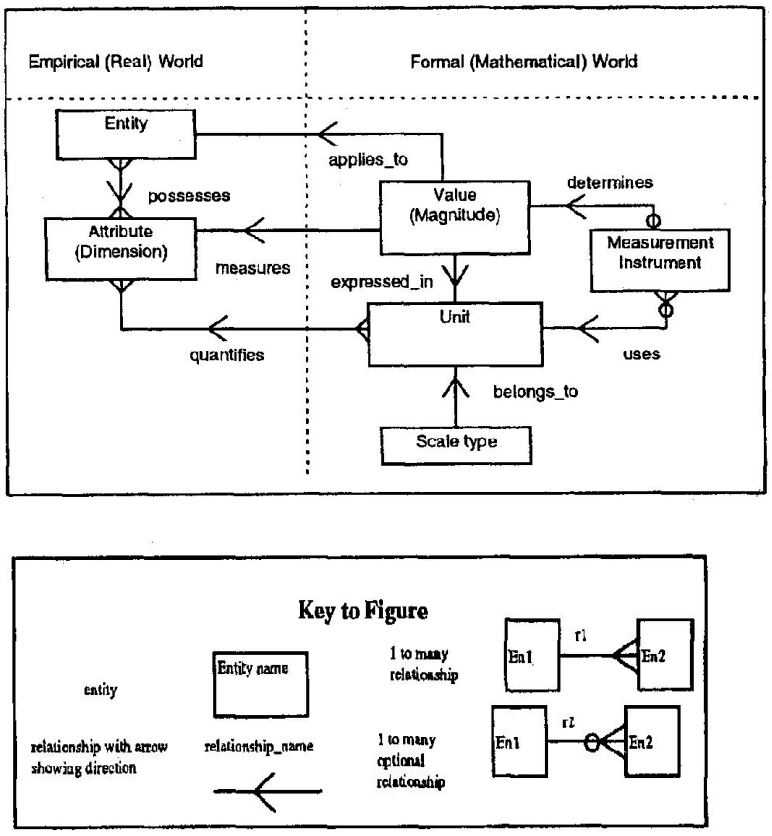

Fig 2.1*: A* structural model of measurement and their notations [KitchenhamPF, 95]

Entities
Entities are the objects, which can be observed in the real world. One of the goals of measurement is to capture their characteristics and manipulate them in a formal way. Software entities may be product as source code, processes as design phase, or resources as people of different types.

Attribute
Attributes are the properties which entities posses, or in other words, attributes define the properties of an object. For instance an attribute of source code is size.
An attribute may apply to one or more different entities. Similarly an entity possesses many attributes. Relationship between them is "many to many" relationship.
There are two types of attributes:

1. Internal attribute
2. External attribute

Internal attribute of product, process or resource are those that can be measured purely in terms of product, process or resource itself. In other words, an internal attribute can be

measured by examining the product, process or resource on its own, separate from its behavior.
External attribute of product, process or resource are those that can be measured with respect to how the product, process or resource relates to its environment. Here the behavior of product, process or resource is important, rather than the entity itself [FentonP, 98].

Units
A measure maps an empirical world to the formal world. A measurement unit determines how an attribute is measured and it is possible to measure an attribute in one or more units. For example, units of distance are “meter” or “kilometer”.

Scale types
Scale type also enables to decide that statement about measurement is meaningful or not. As we can see in table 2.1, there are five types of scale types and it shows that there is one-to-one relationship between units and scale type.

Scale Types	**Basic Empirical Operations**	**Examples**
Nominal	Determination of equality	Mode and Frequency
Ordinal	Equivalence, greater than	Mode, Frequency, Median and Percentile
Interval	Equivalence, Greater than, known ratio for any interval	Relative time, temperature, Intelligence tests
Ratio	Equivalence, Greater than, Known ratio for any interval	Time interval, length, temperature
Absolute	Determination of equality with values obtained from other scales of the same type	Counting entities

Table 2.1: Scale of measurement (refined) [FentonP, 98]

Direct and Indirect measurement
According to Fenton and Pfleeger [FentonP, 98], software measurement is done in two ways:

1. Direct measurement.
2. Indirect measurement.

Direct measurement refers to measuring exactly the object that you're looking to measure. Example of direct measures of the product include lines of code (LOC) produced, errors in source code, memory size, execution speed, etc. For direct measurement, there are certain properties [KitchenhamPF, 95], which must be satisfied.

1. For an attribute to be measurable, it must allow different entities to be distinguished from one another.
2. A valid measure must obey the representation condition.
3. Each unit of an attribute contributing to a valid measure is equivalent.
4. Different entities can have the same attribute value (within the limits of measurement error).

The structural model in Figure (2.1) was presented for direct measurement.
Indirect measurement means that you're measuring something by measuring something else. The measure "module defect density" for instance is calculated by using two other measures "number of defects" and "module size" and both measures are direct measures. Or "productivity" can be calculated by counting the lines of code (LOC) divided by time in months.
Examples of indirect measures include program productivity, program stability, cost of change to requirements etc.
In structural model for indirect measurement (Figure 2.2), attribute association is also mentioned.

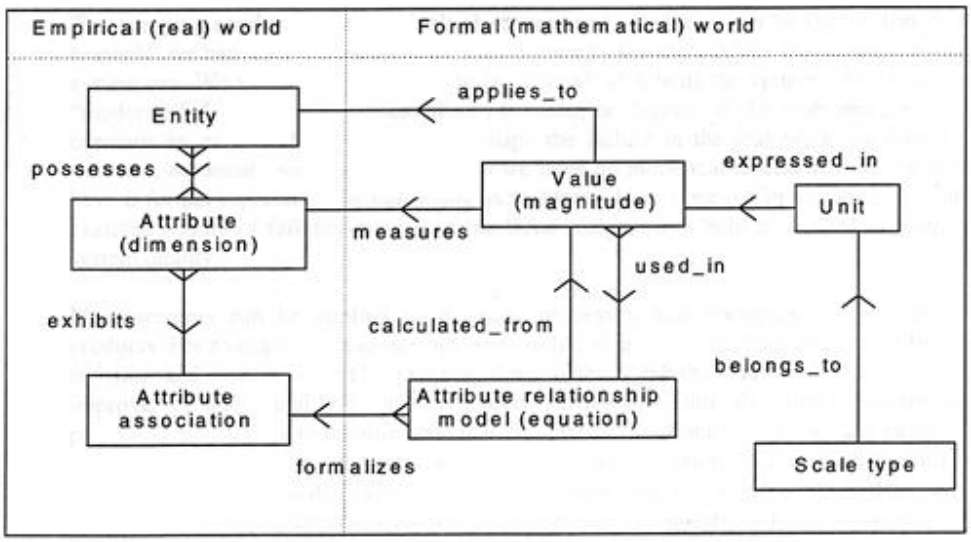

Figure 2.2: A structural model for indirect measures of attribute [KitchenhamPF, 95]

The relation between entities and attributes are many to many. Value applies to entity and value measures the attribute.
Indirect measures should satisfy the following properties [KitchenhamPF, 95]:

1. Measures are based on a model concerning the relationship among attributes defined on specific abstract entities.
2. Measures are based on dimensionally consistent model.
3. Exhibit no unexpected discontinuities, i.e., they should be defined in all expected situations. Like

Measure1= Count1 / Count2 and may present problem if Count2 =0
Measure1= Count1 / Count2-n may be invalid if Count2= n

4. Use units and scales properly. These units and scale types were discussed earlier in this section.

2.4 Validation of measurement

As software industry is moving towards a new era, quality aspects of software development become crucial. Software measurement plays an increasingly important role in understanding and controlling software development practices and products. Numerous software measures have been introduced. These are designed to measure a wide range of attributes, and they naturally vary greatly in definition and in use. Therefore, it is necessary that the measures should be valid. Validity of measures ensures that measures are representing accurately those attributes they purport to quantify. Software measurement validation is a critical way used to assure the quality of software measurements. It also increases the usability and reliability of measurements.
Fenton and Pfleeger defined measure validation as:

"*Validating a software measure is the process of ensuring that the measure is a proper numerical characterization of the claimed attribute by showing that the representation condition is satisfied*" [FentonP, 98].

Representation condition is the rule through which we can map the empirical and numerical worlds and preserving the relations.

2.5 Types of measures validation

According to Kitchenham et al. [KitchenhamPF, 95] there are two types of measures validations:

1. Theoretical validation
2. Empirical validation

Theoretical validation (also called internal validation by [FentonP, 98]) is a basic form of validation and a prerequisite to the empirical validation. Theoretical validation is a theoretical exercise that ensures the metric does not violate any necessary properties of the elements of measurement and confirms a proper numeric characterization of the property it claims to measure. Demonstrating that a measurements measure what it purports to measure is a form of theoretical validation.
Empirical validation (also called external validation by [FentonP, 98]) confirms that measured values of attributes are consistent with predicted values. In theoretical validation, it is ascertained what software attributes are being measured and how to measure those attributes. Empirical validation proves that the measure is useful. It corroborates that measured values of attributes are consistent with values predicted.

In this report we will apply theoretical validation to requirements management measures because it is a basic prerequisite of empirical validation.

2.6 Theoretical validation procedure

In theoretical validation procedure, the first requirement is that the software analyst must have an excellent instinctive understanding of the concept that is being measured. This means that a measure μ of an attribute must be consistent with the intuitive knowledge of this attribute. Theoretical validation then involves modeling of this intuitive understanding of the attribute, which have to be measured [BriandEKM, 95].
Theoretical validation is based on construction of an empirical relation system and formal relation system. Then we have to construct a mapping of these two relation systems in such a way that, properties of empirical relation system should be preserved by the formal relation system if the measure is valid. Fenton and Pfleeger ([FentonP, 98]) delineated the "key stages of formal measurement". And these are used for producing valid software measures.

1. Identify attributes for some real world entities.
2. Identify empirical relations for attributes.
3. Identify numerical relations corresponding to each empirical relation
4. Define mapping from real world entities to numbers.
5. Check that numerical relations preserve and are preserved by empirical relations.

According to the first property we have to identify attributes for some real world entities. For instance, "software requirements specification" (SRS) is entity and an internal attribute of this entity is its size. By following the second step, relations for the attribute size can be "smaller", "bigger" and "equal to". After this, numerical relations corresponding to each empirical relation have to be identified. For example <, >, = can be used for "smaller", "bigger", "equal to". In stage four, mapping from real world entities to numbers have to be done: For that purpose numbers, integers, and real numbers, symbol, etc can be used. Here mapping rule should be defined for mapping, for instance, count only atomic requirements (atomic requirements can be defined as stand alone requirements). In last step, we have to verify the representation condition empirically. Like in our example we can compare empirically size of two SRSs (A, B) and then total number of requirements of these two SRSs. If size of A is greater then B then total number of requirements of A is greater then B.
By following the "key stages of formal measurement" still we cannot say that measures are valid because it is not possible to prove theoretically that numerical relations are preserved by empirical relations, this can be done only empirically. To validate the measures we apply theoretical validation framework proposed by Kitchenham et al. [KitchenhamPF, 95] in which theoretical validation criteria is identified. This framework is general and not specific to particular measures and has been described in section 4.1.

2.6.1 Empirical Relation System

Empirical relation depends upon the attributes that are being measured. For instance, in "size of change to requirement", the relation "less than" can be used. It is important to notice that an empirical relation system does not contain any reference to measures or numbers.
For empirical relations system, understanding of attributes that are being measured is crucial in driving valid and useful measures. However, confusion in terminologies and lack of consensus in the software engineering community make it more difficult to formalize empirical relation system [KitchenhamPF, 95]. An example of an empirical relation system is: "size of change to requirement1" is less than "size of change to requirement2".

2.6.2 Formal Relation system

After the empirical relation system is defined, formal relation system is created. Each entity in the set of entities of the empirical relation system is associated with values. These values have to satisfy the empirical relations. Therefore, the empirical relations are mapped onto relations between the values of measures. For instance, we can assign a value of 30 to "size of change to requirement1" and a value of 40 to "size of change to requirement2". The relation "less than" is replaced in numerical world by "<", and the size of changes is replaced by "number of changes to requirements".

2.6.3 Mapping of Empirical to Formal Relation System

Each relation in empirical relation system corresponds to an element in a formal system. The mapping is done to preserve the relation. The representation condition asserts that a measurement mapping M must map entities in to numbers and empirical relations in to numerical relations in such a way that empirical relation preserve and is preserved by numerical relations.

Empirical world		**Formal world**
less than	mapping	<
equal to	mapping	=
bigger than	mapping	<
double	mapping	*2

Table2.2: Mapping of empirical and formal relation systems

3 Requirement management measures

3.1 Requirement Engineering

What is a requirement? Requirement is need of customer. It can be explained like: requirement is a "condition or capability to which the system under construction (development) must conform". And requirement engineering is defined in [Easter, 04] as:

"Requirement engineering (RE) is a set of activities concerned with Identifying and communicating the purpose of a software-intensive system, and the context in which it will be used. Hence, RE acts as the bridges between the real world needs of user, customer, and other constituencies affected by a software system, and the capabilities and opportunities affected by software-intensive technologies" [Easter, 04].

"Requirement" implies that there is someone who is "requiring", a specific customer who knows what he/she wants. In some projects, requirements are understood to be the list of features (functions, properties, constraints, etc) demands of the customer.
The term "engineering," suggests that requirement engineering is an engineering discipline in its own right, where it is, at most, a fragment of a larger process of software engineering. The term "engineering" also suggests that the outputs of a requirement engineering process need to be carefully engineered, where those "outputs" are usually understood to be detailed specifications [Easter, 04].
Requirements are capabilities and purposes to which any product or service must conform and are common to all development and other engineering activities. To highlight the importance of requirement engineering in software development, the Standish Group conducted a series of surveys of software development projects in the US. They observed the failure rates and identified their critical factors. In the initial study, in 1994, they founded that only 16% of projects were successful and 31% of projects are cancelled before completion. As a whole, $81 billion would spend on cancelled projects by American companies and government agencies in 1995. A follow-up study in 1998 found a small improvement: successful factor rises 16% to 26% and rate of cancelled projects decreased 31% to 28%. At the end, they listed the success and failure factors. Among causes of failure or success, respondents to the survey cited the top three success factors as in [Standish, 94]:

1. User involvement
2. Executive management support
3. A clear statement of requirements

And top three factors leading to failure were:

1. Lack of user input.
2. Incomplete requirement and specifications
3. Changing requirements and specifications

Each above-mentioned factor that leads to failure is related to sub phase of requirement engineering, which is requirement management. ([Easter, 04])
According to [Pressman, 01], the requirement engineering process is described in six distinct steps:

1. Requirement elicitation.
2. Requirement analysis and negotiation.
3. Requirement specification.
4. System modeling.
5. Requirement validation.
6. Requirement management.

This report deals only with step 6.

3.2 Requirement Management

Requirements management can be defined as follows:

"Requirement management is a set of activities that helps the project team to identify, control, and track requirements and changes to requirements at any time as project proceeds" [Pressman, 01].

Requirements management is also Key Process Area (KPA) of "software capability maturity model" (SW-CMM) level 2. SW-CMM was developed by Software engineering institute having focus on improvement in processes of software development [SEI, 93]. Requirement management includes establishing and maintaining agreement between customer and developer on both technical and non-technical requirements. This agreement forms the basis for a set of activities to identify, control, track requirements and changes to requirements at any time throughout the project and for maintaining and enhancing developed software. Important activities include:

- Planning the requirements phase
- Establishing the requirements process
- Controlling requirements changes
- Minimizing the addition of new requirements (scope creep)
- Tracking progress
- Resolving issues with customers and developers
- Holding requirements reviews

Some models, such as the CMMI (Capability Maturity Model Integration) [SEI, 93], put some of these activities in other process areas such as requirements engineering, project management, or configuration management. Requirements management starts with the definition of requirements and continues through the project, culminating in the verification of the end product against the specified requirements [Ludwig, 02].
Requirement management ensures that iterative changes are managed during the project life cycle, considering the overall quality of the product. Requirements management has a

strong connection with quality management in ascertaining what the customer wants (quality). To get high quality in the final product measures are used. Requirements management strongly connected to change management and one core entity of requirement management is requirement change management. Change management is an important yet often problematic part of the software development lifecycle. Even with the substantial knowledge of a system, managing requirement change is by no means straightforward [NASA, 95].

3.3 38 Requirement Management Measures

Many software organizations collect a focused set of metrics on their projects. These metrics provide insight into the size of the product, the effort and time the project or individual tasks consume, and the product quality. Since requirements are an essential project component, measures should be applied to several aspects of requirements management activities to help in judging the state of the requirements and the project.
Software measures used in requirement management process provides the information, required to make key project decisions and to take appropriate actions. Measures are used for getting improvement and control in requirement management process. Metrics should simple to help, keep the project on track and make future projects even more successful. In [Loconsole, 01], a general and comprehensive set of 38-requirement management measures is defined for implementation of the goals of the requirement management KPA (Key Process Area) within the SW-CMM level2.

	Direct/ Indirect	**Measure**	**Scale**	**Entity**
1	Indirect	# Affected groups and individuals about NOC	Ratio	Change request management process
2	Direct	# Base lined requirements.	Ratio	Baseline management process
3	Direct	# Changes per requirement.	Ratio	Change request management process
4	Direct	# Changes to requirements approved.	Ratio	Change request management process
5	Direct	# Changes to requirements incorporated into base line.	Ratio	Change request management process
6	Direct	# Changes to requirements open.	Ratio	Change request management process
7	Indirect	# Changes to requirements per unit of time.	Ratio	Change request management process
8	Direct	# Changes to requirements proposed.	Ratio	Change request management process
9	Direct	# Changes to requirements rejected.	Ratio	Change request management process
10	Direct	# Documents affected by a change.	Ratio	Change request management process, Requirements documentation process
11	Direct	# Final requirements.	Ratio	Requirement management process
12	Direct	# Incomplete requirements.	Ratio	Requirement management process
13	Direct	# Inconsistencies.	Ratio	Requirement management process
14	Direct	# Inconsistent requirements.	Ratio	Requirement management process
15	Direct	# Initial requirements.	Ratio	Requirement management process
16	Direct	# Missing requirements.	Ratio	Requirement management process
17	Direct	# Requirements affected by a change.	Ratio	Requirement management process

18	Indirect	# Requirements schedule for each software build or release.	Ratio	Requirement management process
19	Direct	# TBDs in requirements specifications	Ratio	Requirements specification
20	Indirect	# TBDs per unit of time	Ratio	Requirements specification
21	Direct	# Test case per requirement	Ratio	Requirement management process
22	Indirect	Cost of change to requirement	Ratio	Change request management
23	Indirect	Effort expended on requirements management activity	Ratio	Requirement management process
24	Direct	Kind of documentation	Nominal	Requirements specification process
25	Direct	Major source of request for a change to requirements	Nominal	Change request management
26	Direct	Notification of change (NOC) shall be documented and distributed as a key communication document.	Nominal	Change request management, Requirements documentation process
27	Direct	Phase when change requirements are base lined.	Ordinal	Baseline management process
28	Direct	Phase where change was requested.	Ordinal	Change request management
29	Direct	Reason of change to requirement.	Nominal	Requirement management process
30	Direct	Requirement type for each change to requirements	Nominal	Change request management
31	Direct	Size of a change to requirements.	Ratio	Change request management
32	Direct	Size of requirements	Ratio	Requirement management process
33	Direct	Status of each requirement.	Nominal	Requirement management process
34	Direct	Status of software plans, work products, and activities.	Nominal	Requirement management process
35	Indirect	The computer software configuration item(s) (CSCI) affected by a change to requirements	Nominal	Change request management, Requirement management process
36	Indirect	Time spent in upgrading	Ratio	Requirement management process
37	Direct	Total # Requirements	Ratio	Requirement management process
38	Direct	Type of change to requirements	Nominal	Change request management

Table 3.1: 38 Requirements management measures [Loconsole, 01]

The measures in table 3.1 are defined for better control over requirement management activity. In requirement management different activities are involved like: change request management, requirements documentation, changing cost management, etc. For each activity measures are defined. The goal of this thesis is to validate eight of these measures. One-way we will validate is by performing interviews among the developers to know their views of the measures. Another way is to apply the theoretical/empirical validation techniques, which have described in chapter 2.

3.4 Short survey on requirement management measures

Interviews are mostly helpful for getting the information behind participant's experiences. The aim of interview was to know the opinion and experience of software developers about requirement management measures in order to validate the measures. The plan decides "how" a survey is conducted and the design of the survey is decided in this phase. According to our plan, a set of seven questions was composed (see table 3.2). While composing these questions, the main intention was to extract maximum information through these comprehensive questions. The questions were based on some measures among the "set of 38 requirement management measures". The whole set was also given to software developers, to allow them to answer question number seven.
The approach chosen for the interview was "open ended" and "standardized". This is the kind of interview where questions were asked to all participants and they were free to choose how to answer the questions, they were not bounded only to give answer as "Yes" or "No" [Namara, 99].
Some assumptions on this survey are mentioned below. For instance:

1. This survey is conducted among developers, who were in small number of six. They are in small and medium software development organizations.
2. Change request should be regarded as average change request.

1: What is your opinion and experience about the average rate of change request in development process?
2: What is the average rejection rate of change requests?
3: Normally at which phase maximum changes are requested?
4: Usually what type of change to requirements is requested?
5: What are frequent reasons for change request?
6: What are the major sources of change requests?
7: Do you practice any of above mentioned requirement management measures?

Table 3.2: Set of interview questions

3.5 Result of the survey

While achieving this objective we have interviewed to fifteen software developers by different means like head to head interviews, telephonic interviews, and on electronic mail views were also obtained but we have got partially answers from most of them. Six software developers could answer complete. Brief description of these six software developers and their summarized answers are shown in appendix A.
1. In response of first question, the participants have different opinions. According to them change requests can come within schedule time or after schedule. On medium and small scale, average change request is 45%, it means change is requested in 45% requirements. Change could be requested from any stakeholder like: management team, user, analyst etc. Some developers could not give the exact answer in percentage, they told rate of change requests is higher after the product delivery. Additional change

requests in nonfunctional requirements are higher after the delivery of product (nonfunctional requirements are any requirements that are not about strict functionality, like standards, contractual obligations, external interfaces, performance requirements, useability, robustness, maintainability, etc).

2. This question was difficult to answer for developers. Some of them replied that when a new change requests arrives, screening of that change request is to be done. The requirements, which are approved from screening, are evaluated. The assessment is done while keeping in mind the effort needs on this change request and market point of view. After the assessment, the construction phase begins, in which the evaluated change request is designed and implemented. Most requirements are rejected in screening phase. The percentage of rejection in change requests is maximum in screening phase compared to the other phases. In assessment phase, screened requirements can also be rejected but less then in construction phase. All the interviewed developers mentioned that, the rejection rate of change request becomes very high in later stages of development because the cost of change to requirements in later stages is much higher than in early stages.

3. In response of the third question, developers said that to maintain better control on projects especially in terms of cost and time, it is important to keep record of the phase when change request arrive. Four developers replied that after the design phase the highest number of changes is requested when the design is discussed with stakeholders. Two developers said that many change requests arrives in requirement phase. The remaining one software developer claimed that change requests come after the delivery of product.

4. In answering the fourth question almost all developers have the same opinion. They said that in broad-spectrum, change requests could be distinguished in three categories: addition, deletion and scope modification.

5. In answering the fifth question, half of the participants said that the major reason of change request is to add some functional requirements and remaining half replied that condition tests for error handling (like, developing some exceptions) or to change the logic are important reasons to change requests.

6. In answering the sixth question the developers spent little time. All developers believe that the major source of changes is the customer. The customer can be an individual person.

7. The last question was about practicing the measures from 38 requirement management measures (in Table 3.1). This question was very controversial. The majority of people replied that they are not practicing exactly on some defined measures.
By observing the list in Table 3.1, the interviewed developers answered that the most interesting measures to follow are # Final requirements, # Test case per requirement, Kind of documentation, Reason of change to requirement, Status of each requirement and Total # Requirements (where the symbol “#” means “ numbers”).

3.6 Conclusion

This survey was conducted among developers belonging to small and medium size software development organizations. Normally these organizations are lower than level four of CMM and these organizations are not mature enough to have a measurement program running. Most developers did answer only some of the questions among the provided set. According to them it was rather hard to answer about certain issues on which they have not paid intention, for instance: average rejection rate, frequent reason to change request. Therefore, we acquired only six software developers who had given answers of all questions. Many companies have their own defined measures for requirement management, which are considered as confidential.
The survey conducted was small, we interviewed fifteen developers and six of them could answer completely. Therefore the survey does not have statistical relevance and cannot be used to validate measures. However we can draw some conclusion. From the views and opinion of software developers it is concluded that requirement management measures play a vital role in software development. Most software developers are using measures defined according to the needs of their organization. According to some of software developers, certain measures are rather hard to apply for instance measures, Cost of change to requirement, Effort expended on requirements management activity, The computer software configuration item(s) (CSCI) affected by a change to requirements, Time spent in upgrading.

4 Theoretical validation of a subset of 38-requirement management measures

4.1 Literature review on theoretical validation

Software measurement plays a significant role in understanding and controlling software development process. Along this it is necessary that software measures should be valid. Therefore software validation is critical to the success of software measurement. Many software measures are available in the measurement literature but only few of them have been validated. Software measures are validated in different ways. Basili et al [BasiliM, 00] defined theoretical validation as follows:

1. Analyzing, if a metric is theoretically sound.
2. Verify if a metric fulfill the properties that are associated with the attribute, which is supposed to measure.

Fenton and Pfleeger [FentonP, 98] proposed that domain knowledge is vital before validating the measures and the intuitive understanding of the attribute is preserved when it is mapped to numeric relation system (see figure 4.1). Similarly Melton et al [MeltonGBB, 90] defined five assumptions those mainly concentrated on the representation condition of measurement theory.
Weyuker [Weyuker, 88] defined a set of nine properties for software measures and she suggested that by applying a set of desirable properties that measures should posses we validate software complexity measures. Her work focused mainly on software complexity measures. Basili et al [BasiliBM, 96] also concentrated on property based software measurement and attempted to define a mathematical framework in which they tried to clarify the essential properties for basic concept of software measurement (e.g., size, length, complexity, etc) that are also a common place in measurement validation. Tian and Zelkowitz [TianZ, 92] introduced five axioms, with the goal of better evaluation of complexity measures. They proposed also classification scheme based on additional program characteristics that identify important measures categories. In their approach, programs are represented by means of their abstract syntax trees (e.g., parse trees). Lakshmanian et al. [LakshmananJS, 91] have attempted to define necessary properties for software complexity measures based on control flow graphs. Melton et al. [MeltonGBB, 90] tried to provide a theoretical basis for software document measures validation, which software document measures ought to satisfy.
Most of the works toward software validation have centered on a specific artifact (software complexity measures, document measures, size measures, etc.). Diversity of these works creates problem for researchers and practitioners. It is not clear which approach actually lead to a widely accepted view of validity. To overcome this problem Kitchenham et al. [KitchenhamPF, 95] proposed a theoretical validation framework in which theoretical validation criteria is identified and compared with Melton and Weyuker works. This framework is general and not specific to a particular kind of measures.

In this report, framework proposed by Barbara et al. [KitchenhamPF, 95]. We will apply the key stages of formal measurement of Fenton and Pfleeger [FentonP, 98] to validate eight-requirement management measures chosen from the set of set of 38 requirement management measures defined in [Loconsole, 01].
Premise of theoretical validation procedure is to show that a measure is measuring the concept it is purports to measure. For applying theoretical validation procedure, analyst must have an excellent intuitive understanding of the concept that has to be measured. Measure of an attribute must be consistent with the intuition knowledge of that attribute. Theoretical validation also includes modeling of this intuition knowledge of the attribute, which we have to measure. For theoretical validation procedures it is important to establish an empirical relation system, a suitable formal relation system and a mapping of these two systems in such a way that empirical relation system must be preserved by formal relation system [BriandEKM, 95].

Key stages of formal measurement: [FentonP, 98]

1. Identify attributes for some real world entities.
2. Identify empirical relation for the attribute.
3. Identify numerical relations corresponding to each empirical relation.
4. Define mapping from real world entities to number.
5. Check that numerical relations preserve and preserved by empirical relations.

Theoretical Validation: [KitchenhamPF, 95]

6. For an attribute to be measurable, it must allow different entities to be distinguished from one another.
7. A valid measure must obey the representation condition.
8. Each unit of an attribute contributing to a valid measure is equivalent.
9. Different entities can have the same attribute value.

Figure 4.1: Measures validation Properties

4.2 Application of theoretical validation to eight requirement management measures

With the first two properties of Figure 4.1 we construct the empirical relation system and by means of property 3 we construct mathematical (Formal) relation system. Through property 4 we map these two relation systems, considering empirical relation system as domain and formal relation system as range. The rules of mapping are usually context dependent i.e. depends on the way requirements are collected and documented. However we will suggest some rules for each measure.
Properties 1, 2, 3 (shown in Table 4.1) correspond to the definition of attributes, empirical and numerical relations. Domain, range and scale of each measure are needed for property 4.The *domain* of a function is the set of 'input' numbers for which the

function is defined. The *range* of a function is the set of results or solutions to the equation for a given input. A true function only has one result for every domain. And *function* can be said that in a valid function, domain (x) and Range (y) have a one to one correspondance so that every given range domain value has one and only one range value as a result, but not necessarily vice versa [WikiBooks, 03].
Property 5 and 7 are equivalent and they correspond to the representation condition. Properties 6 and 9 are intuitively satisfied for the eight measures that will be theoretically validated in the next paragraph. Therefore we will not mention these properties in the validation, while property eight is connected to the definition of the unit and the measurement scale, which is done in table 2.1.
In [LoconsoleB, 03] ten measures had been validated among the set of 38 measures defined in [Loconsole, 01]. These measures were "total number of requirements", "number of initial, current, final requirements", "status of requirements", "number of changes per requirements", "status of changes to requirements", "type of change to requirements", "reason for change to requirements" and "cost of changes to requirements".
Prolonging of that work, in this thesis eight measures from the set of 38 requirement management measures will be validated by applying the theoretical validation procedure and the key stages of formal measurement. These measures are "number of final requirements", "number of incomplete requirements", "number of inconsistent requirements", "number of missing requirements", "number of changes to requirements proposed", "number of changes to requirements rejected", "requirement type for each change to requirements " and " size of a change to requirements". These measures will be described in more details in the next sections. Seven of these measures are connected to the attribute "size" therefore the validation for them is very similar to each other. The differences between these seven measures will be in the rationale (the description of the measures) and in the terms of mapping.
Remaining twenty measures are left to validate and this will be done in future work.

4.2.1 Requirements Management process

As we have written in chapter two, measures have to be associated to entities. However we will focus our attention on two entities: requirements management process and change requests management. Requirements management process comprises several activities. These activities include planning the requirements phase, controlling change requests, and requirement documentation and track progress of all activities. These activities can be seen as different sub entities of the requirements management process like change request management, requirement specification, and functions specification. These entities can be measured separately, Table 2.1 shows that among these 38 requirements management measures, 16 of them belong to the entity (requirement management process).
The requirements management process has its focus on change management and on keeping track of the status of requirements. Questions appeared during the requirement management processes are the following: how many are final requirements? How many requirements are missing? How much effort spends implementing each approved

requirement? How many are inconsistent with software product? And record the time that software development team spends on requirements engineering activities.
Four measures related to this entity have been validated in [LoconsoleB, 03], and in this report four measures are validated.

4.2.1.1 Number of final requirements

The measure number of final requirements is used to count the total number of requirements in requirements specification. Requirements specification is a document that is called a system specification if it deals with hardware and software, software requirements specification if it deals only with software. This document is outcome of Analysis & Negotiation phase and which form a satisfactory basis for proceeding to the preliminary design phase. In the design process, the software requirements specification is translated into a logical and physical representation of the software to be implemented. This measure helps in showing product size by knowing the total number of final requirements. For validating this measure we go through each of the properties shown in figure 4.1.

Property1 (identify attribute)**:** The attribute, which has been chosen, is "size of final software requirement specification (SRS)". The reason for choosing this attribute is because time and effort is calculated to some extent from "size of final requirements" because requirements changes are made only to baseline final requirements. This attribute has also relationship with cost estimation.

Property2 (define empirical relations for selected attribute)**:** bigger, smaller, equal to, etc.

Property3 (define numerical relations for selected attribute)**:** >, <, =, etc.

Property4 (define domain and range for mapping from real world entities to numbers)**:** Domain in this case is defined as all the set of software requirements specifications (SRS), while natural numbers or integers can be used as range.
Rule of mapping is defined as no requirement is left deferred (or to be done) and long term-deferred requirements should be rejected. (like pending and deferred requirements should be finalized).

Property 5 and 7 (define representation condition)**:** For representation condition, we have to prove that, if there are two SRS's A and B and size of A is bigger then B then number of final requirements of A is greater then B.

If Size of final (SRS_A)> Size of final (SRS_B)
Then # final requirements (SRS_A)> # final requirements (SRS_B)

4.2.1.2 Number of incomplete requirements

The total number of incomplete requirements can be used to help regarding the capabilities and quality of the delivered product and the project's cost. Especially by knowing the number of incomplete requirements, project schedule can be reorganized

because it connects the scope, work estimates, effort and deadline into a network of software engineering tasks.

Property 1 (identify attribute)**:** "Size of incomplete requirements of SRS" can be used as attribute for formal measurement of this measure.

Property 2 (define empirical relations for selected attribute)**:** bigger, smaller, equal to, etc.

Property 3 (define numerical relations for selected attribute)**:** >, <, =, etc.

Property 4 (define domain and range for mapping from real world entities to numbers)**:** Natural numbers or integers can be used as range while domain in this case is defined as all the set of software requirements specifications (SRS).

As a rule of mapping is "pending requirements" are considered as "incomplete requirements".

Property 5 and 7 (define representation condition)**:** To verify the representation condition we have to demonstrate that given two SRS's A and B, if the size of requirements of A is bigger then B then number of incomplete requirements of A is greater then B.

If Size of incomplete requirements (SRS_A)> Size of incomplete requirements (SRS_B)

Then # Incomplete requirements (SRS_A) > # Incomplete requirements (SRS_B)

4.2.1.3 Number of inconsistent requirements

The description of the product to be developed as given in the SRS is a realistic basis from which inconsistencies can be verified. From this measure it can be determined extra effort and time needed to fulfill these requirements, which are not met in software product. Inconsistency can manifest itself in a number of ways.

- Conflicting terms: two terms having the different contexts to mean the same thing. The term "prompt" to denote a message to have a user input data is used in one requirement while the term "cue" is used by another requirement to mean the same thing.
- Conflicting characteristics: two requirements in the SRS require the software to exhibit contradictory attributes. For example, one-requirement states all inputs shall be via a menu interface while other states all inputs shall be via a command language.
- Temporal or logical inconsistency: two parts of the SRS might specify conflicting timing characteristics or logic. For example, one requirement may state that system A will occur only while system B is running and another requirement may conflict by stating that system A will occur 15 seconds after the start of system B. A logic inconsistency may be one requirement stating that the software will multiply the user inputs, another requirement may state that the software will add the user inputs. [CalgaryU, 98]

In requirements validation process, requirements are checked for consistency and traceability. Inconsistent requirements have conflict (semantically) between individual requirement statements and these may arise because multiple conflicting requirements are

embodied in the requirements specification. Ability to handle inconsistent requirements is crucial to the successful development of requirements specifications. Those requirements, which are not cross-referenced to other documents that they relate, will also be considered as untraceable requirements (like design specification, functional specification, technical specification).

Property 1 (identify attribute)**:** "size of inconsistent requirements of SRS".
Property 2(define empirical relations for selected attribute)**:** bigger, smaller, equal to, etc.
Property 3 (define numerical relations for selected attribute) **:** >, <, =, etc.
Property 4 (define domain and range for mapping from real world entities to numbers)**:** Domain in this case is defined as all the set of software requirements specifications (SRS), while natural numbers or integers can be used as range.
Property 5 and 7 (define representation condition)**:** To verify the representation condition it is possible to say that if given two SRS's A and B, if the size of inconsistent requirements of A is bigger then B then number of inconsistent requirements of A is less then B.
If # size of inconsistent requirements (SRS_A)> size of inconsistent requirements (SRS_B)
Then # Inconsistent requirements (SRS_A)> # Inconsistent requirements (SRS_B)

4.2.1.4 Number of missing requirements

The number of missing requirements is a direct measure. The original SRS documents often experience the problems of missing, not available, and hard-to-locate requirements. One of the reasons to this problem is that there is little consideration for missing requirements from multiple viewpoints. Missing requirements lead to errors in software development and usually also prevent these errors from being detected during the testing phase. Like functional testing is based on the requirements, a missing requirement will therefore not be detected. Operational readiness tests sometimes detect the omissions or inaccuracies, but not missing requirements. Along this, the cost of fixing a mistake (such as missing requirements) increases exponentially the later it is found in the software development process.
The measure number of missing requirements helps to estimate the cost (which has to be spent on completing the missing requirements), rescheduling the project (how to meet the deadline? because missing a deadline can be catastrophic).
Requirements are considered as missing requirements, if they appear in SRS but not in the final product. Partial developed requirements could also be considered as missing requirements (missing from the final system), other missing requirements could be requirements agreed with consumer but not included in SRS.

Property 1 (identify attribute)**:** "size of missing requirements of SRS".
Property 2 (define empirical relations for selected attribute)**:** bigger, smaller, equal to, etc.

Property 3 (define numerical relations for selected attribute) **:** >, <, =, etc.
Property 4 (define domain and range for mapping from real world entities to numbers)**:** Natural numbers or integers can be used as range while domain in this case is defined as all the set of software requirements specifications (SRS).
Property 5 and 7 (define representation condition)**:** Similar to "Number of incomplete requirements" which is stated above.

4.2.2 Change request management

Another entity of requirement management is changes request management. Requirements changes should be made only to baselined requirements. Baseline requirements are those requirements agreed by both customers and developers. They are collected in a document (baseline, or baseline requirement doc) which is the one distributed to the designer of the software system. Any change to the base line can be done only through a submission of a change request. The following is a suggested process for requirements change management [GrifthU, 04]:

- The need for a change must be identified
- A change request is submitted.
- The change request must be approved or rejected.
- Approved changes are documented and incorporated into the baseline.
- Submission of a new baseline should involve a formal review.
- Submitted materials should contain sufficient details so that changes can be backed out if necessary

All change requests will be reviewed on a regular basis by the project change control board. The change manager will drive the schedule based on the number and complexity of change requests and a Cost/Schedule Impact Analysis (CSIA) is demanded. The requirement change control team reviews the requested change and either accepts, reject, or defer and also ensure that the resources are neither scarce nor wasted during change activities [GrifthU, 04].
For this important entity 16 measures are defined among 38 requirement management measures. Among these, three measures have been validated in [LoconsoleB, 03].

4.2.2.1 Number of changes to requirements proposed

A change proposal is a document that describes a necessary change. A change request may come from a trouble report or for request to have additional functionality. Whenever any stakeholder (stakeholders are customer, users, domain experts, partners, and software team etc) determines that some aspect of an accepted work product should be changed, then that party submits a change proposal to the Change Control Board. Careful consideration is carried out on these change proposals and an estimate of the effects of implementing the proposed changes is produced. On the behalf of the impact analysis change proposals are accepted or rejected. From this measure stability of requirements is estimated.

Property 1 (identify attribute)**:** "Size of proposed changes to baseline requirements".

Property 2 (define empirical relations for selected attribute)**:** bigger, smaller, equal to, etc.

Property 3 (define numerical relations for selected attribute)**:** >, <, =, etc.

Property 4 (define domain and range for mapping from real world entities to numbers)**:** Domain in this case is defined as all the set of software requirements specifications (SRS), while natural numbers or integers can be used as range.

The rule of mapping can be defined as only those change requests are considered which arrived by stakeholder and that modify the semantic of the requirements (affected).

Property 5 and 7 (define representation condition)**:** To verify the representation condition it is possible to say that if given two SRS's A and B, if the size of proposed changes A is bigger then B then number of changes to requirements proposed of A is greater then B.

If Size of proposed changes (SRS_A)> Size of proposed changes (SRS_B)

Then # Changes to requirements proposed (SRS_A)> # Changes to requirements proposed (SRS_B)

4.2.2.2 Number of changes to requirements rejected

Any proposed change request can be rejected after impact analysis and the status of change request is set to reject. Regardless of whether a change is approved or rejected, the information about change requests are recorded by a defect tracking system and made available to the party submitting the change proposal (and any other interested parties that desire to monitor the progress of the work product). This measure also helps to estimate the stability of requirements.

Property 1 (identify attribute)**:** "Size of rejected changes to proposed change requests".

Property 2 (define empirical relations for selected attribute)**:** bigger, smaller, equal to, etc.

Property 3 (define numerical relations for selected attribute) **:** >, <, =, etc.

Property 4 (define domain and range for mapping from real world entities to numbers)**:** Domain in this case is defined as all the set of software requirements specifications (SRS), while natural numbers or integers can be used as range.

Proposed change requests will be rejected instantly if it determines that the cost of formally assessing the impact of the change outweighs its perceived benefit.

Property 5 and 7 (define representation condition)**:** The representation condition can be verified by comparing the size of rejected change requests of two SRS's A and B. If the size of change requests of A is bigger then

B then number of changes to requirements rejected of A is greater then B.

If Size of change requests (SRS_A)> Size of change request (SRS_B)

Then # changes to requirements rejected (SRS_A) > # changes to requirements rejected (SRS_B)

4.2.2.3 Requirement type for each change to requirements

This measure shows the requirement type for each change to requirements. There are many types of requirements like: System requirements (such a set of requirements will usually refer to the requirements that describe the capabilities of the system with which, through which, and on which a certain product will function), user requirements (user requirements will describe the needs, goals, and tasks of the user), functional requirements (what the product is supposed to do by defining specific functional areas), nonfunctional requirements (performance requirements, constraints and quality attributes) and market requirements.
Basically this measure is used for classification of requirements affected by change. It could also be used in conjunction with other measures for determining the final status of software requirement specification.

Property 1 (identify attribute)**:** "Requirement type category".

Property 2 (define empirical relations for selected attribute)**:** Same, different, etc

Property 3 (define numerical relations for selected attribute)**:** =, ≠, etc.

Property 4 (define domain and range for mapping from real world entities to numbers)**:** The range for this measure is set of system requirements, user requirements, functional requirements, and nonfunctional requirements.
The rule of mapping can be defined as: a requirement must belong to only one of requirement types.

Property 5 and 7 (define representation condition)**:** For representation condition, we have to compare the requirements types of two requirements empirically and numerically. Empirical and formal properties would be preserved by this comparison.

4.2.2.4 Size of a change to requirements

Size of change to requirements narrates the complexity of change to requirements. Requirements are changing fairly constant during a project development therefore it is necessary to control the size of change to requirements. This measure helps to estimate the effort required for that change, based on assigned resources and estimate schedule and cost impact.

Property 1 (identify attribute)**:** "cost of change".

Property 2 (define empirical relations for selected attribute)**:** greater than, smaller then, equal to, etc

Property 3 (define numerical relations for selected attribute) **:** >, <, =, etc.

Property 4 (define domain and range for mapping from real world entities to numbers)**:** Domain in this case is defined as all the set of software requirements specifications (SRS), while natural numbers or integers can be used as range.

Property 5 and 7 (define representation condition)**:** To verify the representation condition it is possible to say that if given two changes to requirements A and B, if cost required to perform A is greater then B then size of A is greater then B.

If cost of change (changes to requirements A)> cost of change (changes to requirements A)

Then Size of (changes to requirements A)> Size of (changes to requirements B)

5 Summary and Conclusions

Measurement is fundamental part of developed sciences. In any field of science, understanding of performance and outcome is developed by quantitative description of key process and products that is created through use of measurement. Requirement management measures allow making the best decisions while trying to meet the needs of end users through identifying and specifying what they need. To use these measures, initiate understanding of environment and goals of measurement is required.
The goal of this dissertation was to validate eight requirement management measures from a set of 38 requirement management measures. The methods used to achieve the goal were:

1. To conduct a survey of software developers to know their experiences and views.
2. To apply theoretical validation procedure to the eight measures

In applying the first method, we have not been so successful to collect massive data and conclude better result from survey. Foremost difficulty encountered is was to gather data of developers both from Europe and Asia. Due to the different trends of development they have quite different vision about measurement. The second problem was that, most developers could answer only some of questions among the provided set. It seems difficult for them to answer some questions because they have not paid attention to some points that were asked, for instance: average rejection rate. That's why it was not remain possible to mention their descriptions in the thesis. Therefore we acquired only six software developers who have given answers of all questions. Along this many companies have their own defined measures for requirement management, which are considered as confidential.
However we can still show some conclusions. From the views of software developers it is concluded that measures are in use to some extent but developers also use their own experiences to performing requirement management activity rather than to follow some measures. There is still need to practice requirement management measures to establish the correctness of this phase.
In applying the second method to validate the eight requirements measures we encountered some difficulties, like it was difficult to know which view of validity is widely accepted and have the consensus of the software community. Secondly, verifying representation condition was difficult because representation condition needs to be verified in the context of an empirical study having real requirements specification but our situation was not same. Therefore in this thesis eight of the measures have been validated. Remaining twenty measures needs to be validated in future work.

References

[BasiliBM, 96] Basili R Victor, Briand C Lionel, Morasca Sandro, *"Property Based Software Engineering Measurement"*, IEEE Transactions on Software Engineering, Vol 22, No 1,pp 68-86, Jan 1996

[BasiliM, 00] Basili R Victor, Mendonca G Manoel, *"Validation of an Approach for improving existing measurement frameworks"*, IEEE Transactions, Software Engineering, Vol 26, No 6, June 2000

[BriandEKM, 95] Lionel Briand, EI Emam Khaled, Morasca Sandro, *"Theoretical and Empirical Validation of Software Products Measures"*, International Software Engineering Research Network Technical Report, #ISERN, 1995

[CalgaryU, 98] Calgary University, SENG 611, *"Requirements Engineering"*, Graduate Course in Software Engineering, Canada, 1998 http://sern.ucalgary.ca/courses/SENG/611/W04/

[Easter, 04] Steve Easterbrook, *"What are Requirements"*, 2004 http://jasonnolan.net/kmd1002/easterbrook.pdf.

[FentonP, 98] Fenton E Norman, and Pfleeger L Shari, *"Software Metrics- A Rigorous & Practical approach"*, 2nd Edition, International Thomson Publishing, Boston, MA, 1998

[GrifthU, 04] Griffith University, Information Technology 2203INT, *"Software Process Management"*, Software Quality Institute, UK, 2004 http://www.int.gu.edu.au/courses/2203int/req

[Kelvin, 89] Lord Kelvin, *"Popular Lectures and Addresses"*, 1889 http://www-history.mcs.st-andrews.ac.uk/history/ Mathematicians /Thomson.html

[KitchenhamPF, 95] Kitchenham Barbra, Pfleeger L Shari and Fenton E Norman, *"Towards a Framework for Software Measurement Validation",* In IEEE Transactions on software Engineering, 21(12), December, 1995

[LakshmananJS, 91] Lakshmanan B Kadathur, Jayaprakash S, and Sinha P, *"Properties of Control-Flow Complexity Measures,"* IEEE Transactions. Software Engineering, vol. 17, no. 12, pp. 1289-1295, Dec. 1991

[Loconsole, 01] Loconsole Annabella, *"Measuring the Requirements Management Key Process Area"*, Proceeding of ESCOM – European Software Control and Metrics Conference, London, UK, April 2001

[LoconsoleB, 03] Loconsole Annabella, Börstler Jürgen, *"Theoretical Validation and Case Study of Requirements Managements Measures"*, ISSN-0348-0542 03, 2003, Umeå University Report, UMINF 03.02

[Ludwig, 02] Ludwig Consulting Services, LLC, *"Managing Requirements"*, 2002 http://www.jiludwig.com/Requirements_Management.html

[MeltonGBB, 90] Melton C Austin, Gustafor A David, Bieman M James, Baker L Albert, *" A Mathematical perspective for software measures research"*, IEEE/BCS Software Engineering Journal 5(5): 246-254,1990

[Namara, 99] Carter McNamara, *"General Guidelines For Conducting Interviews"*, Lecture notes, 1999 http://www.mapnp.org/library/evaluatn/intrview.htm

[NASA, 95] National Aeronautics and Space Administration, Software Engineering Laboratory Series, *"Software Measurement Guidebook"*, Revision 1, 1995 *http://sel.gsfc.nasa.gov/website/documents/online-doc/94-102.pdf*

[PfleegerJCK, 97] Pfleeger L Shari, Jeffery Ross, Curtis Bill and Kitchenham Barbra, *"Status Report on Software Measurement"*, 0740-7459/97, IEEE Software, March/April, 1997

[Pressman, 01] Pressman S Rogers, *"Software Engineering, A Practitioner's Approach"*, Fifth Edition, McGraw Hill Higher Education, 2001

[Roberts, 79] Roberts Fred S, *"Measurement Theory with Applications to Decision making, Utility, and the Social Sciences"*. Addison Wesley Publishing Company, 1979

[SEI, 93] Software Engineering Institute, *"Capability Maturity Model for Software (SW-CMM)"*, 1993. http://www.sei.cmu.edu

[Standish, 94] Standish Group Report International, *"The Chaos Report"*, 1994 http://www.projectsmart.co.uk/docs/chaos_report.pdf.

[TianZ, 92] Tian Jianhui and Zelkowitz V Marvin, *"A Formal Program Complexity Model and Its Application"*, Journal of System and Software, vol. 17, pp: 253-266, 1992

[Weyuker, 88] Weyuker J Elaine, *"Evaluating Software Complexity Measures"*, In IEEE Transactions on Software Engineering, 14(9): 1357-1365, 1988

[WikiBooks, 03] WikiBooks, *"Algebra Functions"*, July 2003. http://en.wikibooks.org/wiki/Algebra:Functions#Functions_in_terms of_Domain_and_Range

[Zuse, 95] Zuse Horst, *"History of Software Measurement"*, 1995 http://www.literateprogramming.com/m3hist.pdf

Appendix A

Interviewing the software developers about the specific questions was one of the major parts of this dissertation. To accomplish this task, five software developers were interviewed. A brief description of them is as below:

Interview 1
Designation: Project manager, Soon Soft, Pakistan
Profile: He has been dealing with IT related projects since last eight years and has worked on different positions ranging from software developer to project manager.

Interview 2
Designation: Software developer, Assiduous technologies, Surat, India
Profile: Specializing in Object-Oriented design and analysis with extensive experience in the full life cycle of the software design process including requirements defining, prototyping, design, interface implementation and maintenance.

Interview 3
Designation: IT Manager, Assiduous technologies, Surat, India
Profile: He has been working with planning, developing and implementing state of the art information solutions facilitating corporate growth.

Interview 4
Designation: Project manager, Assiduous technologies, Surat, India
Profile: He leads cross-functional teams with diverse technical backgrounds. He has worked on different positions ranging from software developer to project manager.

Interview 5
Designation: Software developer, Assiduous technologies, Surat, India
Profile: He has been working with different software development projects since five years.

Interview 6
Designation: System developer, Umeå University, Sweden
Profile: He has a professional experience of software development.

Afore mentioned people were asked the following set of questions.

1: What is your opinion and experience that approximately what is average rate
2: What is average rejection rate of change requests?
3: Normally at which phase maximum changes are requested?
4: Usually what type of change to requirements is requested?
5: What are frequent reasons for change request?
6: What are the major sources of change requests?
7: Do you practice any of above mentioned requirement management measures*?*

Set of Interview Questions

In the light of these questions many answered were gathered and recorded. Some of them are discussed here.

Answers 1: Almost all answers have the same theme as these answers. One developer said: "The rate of change request in medium size development organizations is almost 45% to all requirements but it decreases in small organizations and increases in large scale development organizations". Another developer who could not give exact answer replied "The percentage of change request increases after product or service delivery because the complete product is in use of the customer"

Answers 2: Answers to this question were not precise. Like:
"Rejections are done in three steps. When change request comes, it must go through screening, assessment, and then construction. And rejection gradually decreases with these steps."
"Rejection rate increases in later stages of development cycle due to cost"
"Cost in change request is massive when change request is not be undertaken in early stages"

Answers3: "In design phase, maximum changes are requested"
"In design phase, when design is discuss with customer, then most changes are requested"
"In requirement phase most changes comes"

Answers 4: "addition some thing (functionality) is greater."
"Requests of change can be of addition, deletion or scope modification and addition change is most frequent"

Answers 5: "To add some function"
"When customer operates he/she might needs some more functions to help in operating that product, make it more user friendly."
"Exception handling is frequent change request, due to unexpected behavior of software"

Answers 6: "Customer"
"Customer is major source"

Answers 7: "I am not exactly following any described measures"
"There is no need for measures in small scale development organizations"
"I am working according to experience, but in these measures 11, 21, 24, 37 are such measures which I practice, may be because these measures are included in my experienced knowledge"