Product and Management Metrics for Requirements

Master Thesis

Ganesh Kandula, int04gka@cs.umu.se

Vinay Kumar Sathrasala, int04ksy@cs.umu.se

March 31st 2005

Department of Computing Science
Umeå University
Umeå
Sweden

Abstract

Software requirements are the initial step in a software development cycle. Gathering requirements and managing requirements well are key factors to a successful software project. Many software projects have failed because of poor requirements. Nowadays software requirements is a very hot topic in software industry.

This thesis describes metrics that are useful to the requirements manager as they allow to establish an effective communication between the development team and the customer, to manage the requirements phases effectively and efficiently and to produce a quality end product. Using the metrics proposed in this thesis, the requirements manager can easily improve the management of requirements and set goals for the organization to achieve maximum quality in the end product.

Acknowledgements

We have been fortunate to work under the supervisor Mrs. Annabella Loconsole. We appreciate her constant support and valuable advice from time to time. It is our immense pleasure to express our deep sense of gratitude to her for providing us an opportunity to learn this subject under her guidance.

We also thank Johanna Högberg, who has given us an opportunity to write the thesis under her guidance. She supported us in various ways in our thesis. We express sincere thanks to her, for giving support and suggestions given from time to time.

We thank Mr. Per Lindström, Director of Studies in Computing Science Department at Umeå University. Accepting the thesis and for his guidance in finalization of the thesis.

Last but not least, we wish to extend our thanks to our family members, friends and every one who has contributed even in a small way in completing the thesis report.

INDEX

Contents Page No

1. Introduction
   1.1 Introduction 8
   1.2 Outlines of Thesis 9

2. Software Requirements Specification 10
   2.1 Software Quality 10
   2.2 What is a SRS? 10
   2.3 Attributes of SRS 11

3. Software Metrics 18
   3.1 Why Do We Need Software Metrics? 18
   3.2 Definition of Software Metrics 18
   3.3 Measurement 19
   3.4 Goal/Question/Metrics 20
   3.5 Classification of Software Metrics 21
      3.5.1 Product Metrics 21
      3.5.2 Process Metrics 24
   3.6 Scope of Software Metrics 24

4. Requirement Product Metrics 26
   4.1 Introduction 26
   4.2 Metrics 26
   4.3 NASA Categories 29
      4.3.1 Individual Specifications 29
      4.3.2. Entire Requirement Document 30
   4.4 Automated Requirement Management Tool 31
      4.4.1 ARM Analysis Process 32
      4.4.2 ARM Analysis of Document Set 32
      4.4.3 Software Requirement Specification document 32

5. Requirements Management Metrics 34
   5.1 Introduction 34
   5.2 Why Do We Need Requirements Management? 34
   5.3 Tasks of Requirement Management 35
   5.4 Models for Requirements Management process 38

5.4.1. Coarse-Grain activity models 39
5.4.2. Fine-Grain Model 40
5.4.3. Role-Action Model 41
5.5 Applying GQM Technique to Improve the Requirements Management Process 43

Conclusions 53

Appendix 55

References 61

Chapter 1

1. INTRODUCTION

Requirements are the driving force behind a software projects. Every phase in software development like analysis, design and testing, etc, directly and indirectly depends up on the requirements. That is why nowadays most of the software development organizations are giving importance to requirements, to the software development process and the quality of the software product. Information system projects or software project frequently fail, the failure rate of large projects is reported as being between 50% - 80%. Because of the natural human tending to hide bad news, the real statistic may be even higher [Paul]. Many software products have failed because of poor requirements engineering activities. The reason for this problem is lack of training in software development on the part of developers and project managers, many of whom have undergraduate computer science degrees, but are not good at requirements. So to avoid this problem, every software development organization should train some designers (developers) on requirement engineering.

Software Requirements Specification (SRS) means establishing a contract between the customer and the developer or requirement manager. The SRS gives full details on the customer expectations and our consent to deliver a product that fulfills them. SRS is just like a contract paper, where we will mention which requirements. We are going to include and what the result of the entire product will be. Well documented/written SRS will reduce efforts of the programmer, because they will easily understand the requirements and write correct and efficient code for the project. Well written SRS will also reduce the cost of the project. Many people are doing their research on the requirements area; we are doing our thesis on this topic. We are concentrating on Requirements Product Metrics and Requirements Management Metric. In Requirements Product Metrics, we specify some attributes, which are useful while writing the SRS document. We also specify some techniques to measure these attributes and we specified a matrix, which will be useful to write efficient software requirement specifications. We discuss the Automated Requirement Measurement (ARM) tool, which is helpful in measuring the quality of the SRS. In Requirement Management Metrics, we describe the tasks and models which are helpful in managing the requirements in the requirements phase. We have applied the Goal/Question/Metric to define metrics for the management of requirements.

Outline of Thesis:

Chapter 1: Gives an introduction to the thesis and describes the goals of our thesis work. How we found the problems and how we started our thesis work.

Chapter 2: Describes the software quality, Software Requirement Specifications and their definitions.

Chapter 3: Describes software metrics, the need of software metrics, their measurement and the classifications of software metrics.

Chapter 4: Describes the requirement product metrics and their measurements. We specify a matrix to improve the software requirement specifications (SRS) and introduce the Automated Requirement Measurement (ARM) tool.

Chapter 5: We describe the tasks, models which are helpful to manage the requirements. We also apply the Goal/Question/Metric to define the metrics, needed to improve the requirements management process.

Chapter 2

Software Requirements Specification

2.1 Software Quality

The area of software engineering offers some excellent approaches for measuring software quality. The meaning of the word '*Quality*' varies from person to person; for instance the customer needs quality software, which is bug free. But attaining 100% quality and bug free code is near impossible. Therefore all the software producers try to attain at least a customer appreciation quality of about 100%. Based upon different needs, the quality is given different meanings. There are different approaches to achieve software quality. Freeman states [Oulu94]:

"*Quality should be the driver of the entire development system.*"

Gild also has a similar idea on quality and places emphasis on control of the critical quality and resource attributes and mainly targets on usability and maintainability **[**Oulu94]. Quality is, as mentioned earlier, viewed differently by different people. In the view of the customer quality means the end product is error/bug free one. The developer wants to develop more efficient and reliable program for the project. The Developing Manager needs to develop a project which is efficient, cost effective, and reliable and also customer satisfied. Development of a quality product is the key to customer satisfaction. Software which does not fulfill the pre-requisites is an indication of failure. Measuring the quality of the software is a vital step in software engineering. Quality should be measured from the early stages of software building or else the development can end up with software that fulfills the requirements but fails to satisfy the customer.

2.2 What is a Software Requirements Specification?

Software Requirement Specifications (SRS) is a document that contains the complete external behavior of the software system. It gives the overall structure of the system, e.g. how it will work and what it will output. SRS is the first step in the software development, based on user requirements; developers can estimate the cost and time of the project. The main purpose of the SRS is to:

1. Communicate among customers, users, analysts, and designers
In this phase, those who are involved in the project will discuss with each other and will write down all specifications.

2. Supporting system-testing activities
All requirements should support the system test cases, otherwise there will be problems in the test phase and changing the requirement in this phase, will affect the overall cost of the project in a negative way.

3. Controlling the evolution of the system

SRS should be flexible, then only user can easily modify the SRS document. Engineers are facing some problem with writing good requirement (SRS), because engineers are not trained to write requirements. According to IEEE, requirement and system are defined as:

Requirement: A statement identifying a capability, physical characteristic or quality factor that bounds a product or process need for which a solution will be pursued.

System: The top element of the system architecture, specification tree, or system breakdown structure that is comprised of one or more products and associated life cycle processes and their products and services.

2.3 Attributes of SRS

We consulted some books and IEEE papers, and based on our understanding the nineteen attributes shown in Table 1 are the most important attribute of SRS.

Serial Number	Attribute	Reference
1	Unambiguous	[Davis93], [IEEE93]
2	Correct	
3	Complete	
4	Verifiable	
5	Understandable	
6	Modifiable	
7	Traced	
8	Traceable	
9	Design Independent	
10	Annotated	
11	Concise	
12	Organized	
13	Not Redundant	[IEEE93]
14	Achievable	
15	At Right Level of Detail	
16	Reusable	
17	Electronically Stored	
18	Necessary	[Davis93], [Wiegers99].
19	Consistent	

Table 1: SRS Quality Attributes

1. Unambiguous

Ambiguity in requirement specifications causes numerous problems. *An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation [IEE84]*. Ambiguity occurs, where one word (phrase) has several meanings. If more than one person read the SRS document, they may have different interpretations. To avoid this, one can maintain a glossary.
Example 1.a: *Air Traffic Controller System* [Davis93].
"For up to 12 aircrafts, the small display format shall be used. Otherwise, the large display format shall be used". In the above statement there is an ambiguity by reading. The developer may not know by reading this specific requirement whether to include or exclude 12.

2. Correct

An SRS is correct if and only if every requirement stated therein represents something required of the system to be built. In software systems built nowadays it is hard to write correct SRS, because requirements are changing according to user needs. The Fig 1 explains the definition of correct.

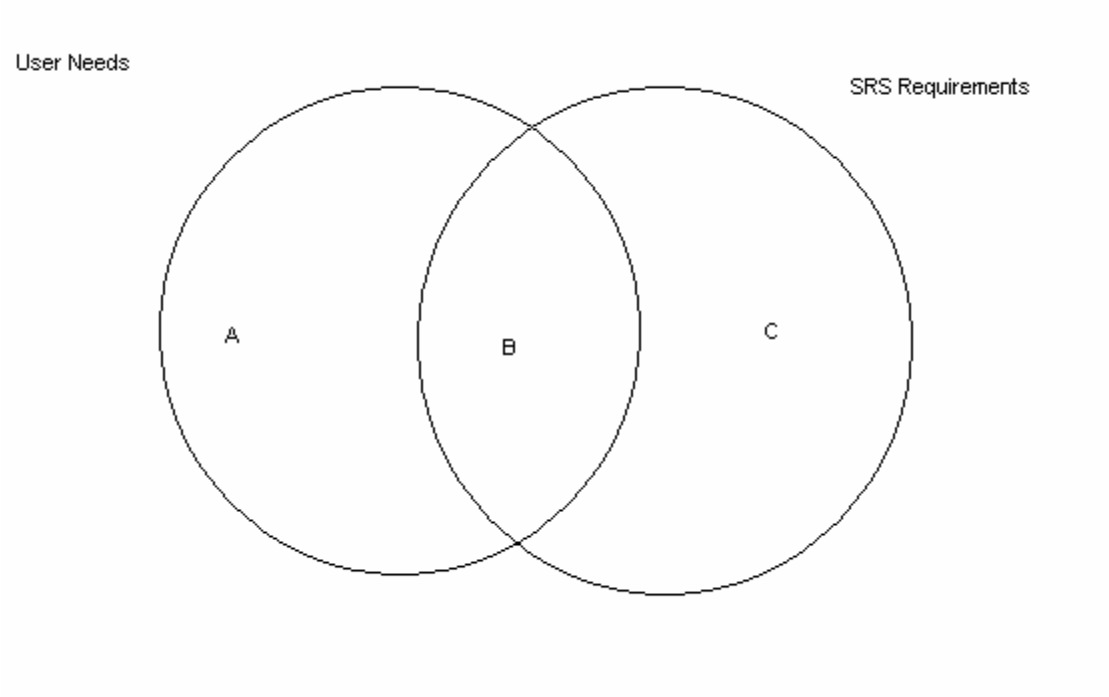

Fig 1: Venn diagram to show the definition of Correct

Circle 'A' represents the User Needs, while circle 'C' represents the SRS Requirements. If an SRS is correct, then the intersection of A and C is equal to A union C. It means that every requirement in the SRS is used in the development of the software (product). The example of Fig.1 represents incorrect SRS.

3. Complete

An SRS is complete if:

- *Everything that the software is supposed to do is included in the SRS* [Davis93]
- It is not possible to write complete and correct SRS. If SRS is complete, then the region of B in Fig1 is equal to A and C, because user needs and SRS requirement are same.
- *All pages numbered; all figures and tables numbered, name referenced; all terms defined; all units of measure provided; and all referenced material present* [IEEE84].
- *No sections marked "To Be Determined"* [Davis93].
- If you include 'TBD' in the SRS it will give the ambiguity to the developers. Instead of putting TBD in SRS, just write the date and the responsible person name, which is easier for all involved.

4. Verifiable

An SRS is verifiable if, and only if, every requirement stated therein is verifiable. *A requirement is verifiable if, and only if, there exists some finite cost effective process with which a person or machine can check that the actual as built software product meets the requirement [IEEE84] [Davis93].* All requirements specified in SRS must be verifiable; it will reduce the cost in test phase. If a requirement is verifiable mainly depends on ambiguity and undecideability. If a requirement written in SRS is ambiguous, then to verify such requirements is very difficult, as discussed in Fig 1. If two requirements have the same meaning, then it is very hard to decide which one to take.

5. Understandable

An SRS should be understandable by every one. Generally some notations used in SRS are not understandable by the non computer users. To avoid this problem we should write SRS document in natural language which is understandable by both customer and developer.

6. Modifiable

An SRS is modifiable if its structure and style are such that any changes can be made easily, completely, and consistently [IEEE84]. SRS should be modifiable, so that the user can easily track the system. This means that the user can change or add other useful requirements to the SRS. Usually we maintain table with index and contents, so that we can easily check the location of the requirement and we can modify the requirement. One technique that can be used to improve the readability of a SRS is to repeat selected requirements in different locations in the document. This attribute of an SRS is called redundancy [Davis 93]. For example, in describing the external interface of a PABX, we must define interaction between the user and the telephone switch, here we have two types of call options, one is local call and other one is long-distance (ISD or STD). For the external view of local call, the requirement is:

Starting with an idle telephone, the user should lift the handset, the system shall respond with a dial tone, and then the user should dial the seven digit phone number of the party the user is trying to reach.
For the external view of long-distance call, the requirement is:
Starting with an idle telephone, the user should lift the handset, the system shall respond with a dial tone, and then the user should dial a 1 followed by the ten-digit phone number of the party the user is trying to reach [Davis93].
In the above example statements are repeated, so the users can easily read the SRS document.

7. Traced

An SRS is traced if and only if the origin of each of its requirements is clear [Davis93].
This means that if we know the location of the requirement in SRS, we can easily monitor the behavior of the requirement.

8. Traceable

There is a difference between Traced and Traceable. *An SRS is traceable if and only if it is written in manner that facilitates the reference of each individual requirement [Davis93].*

There are a variety of effective techniques for achieving traceability [Davis93].

- Number every paragraph hierarchically. Later you can refer by using paragraph and sentence number, e.g., requirement 2.3.2.4s3 refers to the requirement in sentence 3 of paragraph 2.3.2.4.
- Number every paragraph hierarchically and include only one requirement in any paragraph. Later refer the requirement using paragraph number.
- Number every requirement with a unique number in parentheses immediately after the requirement in the SRS.
- Use a convention for indicating a requirement, e.g., always uses the word shall in a sentence containing a requirement; then use a simple shall-extraction tool to extract and number all sentences with shall.

9. Design Independent

An SRS is design independent, if there exists more then one software architecture or system design. Example: if a user wants to develop a game (Cricket), he/she can give all specifications to the developers. If more then one developer designs the system, they will design it in different ways, and the user can choose the best design. If SRS is design dependent then the developers need to stick to the particular software architecture or system design.

10. Annotated

Annotating requirement in an SRS provides guidance to the development organization [Davis93]. Annotating gives a brief description of requirements in SRS to the software developers. There are three types of annotations.

- *Annotated by Relative Importance:* If SRS is annotated by relative importance, the designer can easily understand which requirements are important and which are not important to the customer. This approach satisfies minimum needs of the customer.

- *Annotated by Relative Stability:* If SRS is annotated by relative stability, the designer should know which requirements are changing most likely. Then the designer can easily build the system in that flexibility.

- *Annotated by Version:* If SRS is annotated by version, it tells the reader which requirements have been fulfilled in different stages (versions) of the development.

11. Concise

Given two SRS for the same system, each exhibiting identical levels of all the previously mentioned qualities, the SRS that is shorter is better [CAR90] [Davis93]. If an SRS is concise, then the user can easily read the entire document.

12. Organized

If an SRS is organized, then the user can easily locate the information and understand the logical relation between the different sections. If we maintain a general organization format, it will be easy to write an SRS. Organizing the SRS is like adding an index to a book. For instance the cricket game example, we can organize the format as follows:

- Group the variables into one class
- Group the constants into one class
- Group the players information into one class
- Group the functions into one class

13. Not Redundant

An SRS is redundant if the same requirement is stated more then once. But redundancy increases readability of SRS. In some case redundant requirements can lead to inconsistency problems in SRS. In the example 6, with additional readability comes the potential for decreased modifiability because a later modification to only one occurrence would render the SRS inconsistent [Davis93].

14. Achievable

If a complete system is designed using all SRS requirements, then the SRS is achievable.

15. At Right Level of Detail

The Right level of detail is a function of how the SRS is being used. This gives detailed information to the customer and developers, regarding which requirements have been used in the system. It means that if an SRS specified clearly then the developers can easily understand which variables or functions are used in particular position or situation.

16. Reusable

An SRS is reusable if and only if its sentences, paragraphs and sections can be easily adopted or adapted for use in a subsequent SRS.

17. Electronically Stored

If the requirements are stored in an electronic database then the SRS can be generated automatically from the database.

18. Necessary

The stated requirement is an essential capability, physical characteristic or quality factor of the product or process. If it is removed or deleted, a deficiency will exist, which cannot be fulfilled by other capabilities of product or process.

19. Consistent

A **c**onsistent requirement do not conflict with other software requirements or with higher level requirements. It means it is a unique requirement.

Chapter 3

Software Metrics

A Metric is a measure of quality, metrics can be used to improve software quality and productivity and are used to measure software resources, products and processes. When we collect data for software metrics there should be a description of the data so that it can be easier for software developers to do their job. Software metrics give the overall information about the development product, like cost, time and all phases' information.
Ideal metrics should be [Everald88]:

- Simple
- Objective
- Easily obtainable
- Valid
- Robust

Metrics should measure what they are intended to measure should be understandable to the greatest extent possible, at a reasonable cost.

3.1 Why Do We Need Software Metrics?

Software metrics allow us to better understand the software development environment and give detailed information about the project. Metrics are used to predict, manage, and control the product. With good metrics we can plan the project in a proper way. Gathering data and using the data is quite complicated. If employees are able to understand and write down the requirements well, then it will be easier to develop the product. Software metrics provide guidelines that tell us where we are and where are we going in the software development process.

3.2 Definition of Software Metrics

Software metrics are the building blocks of software engineering. Goodman defines them as "*the continuous application of measurement-based techniques to the software development process and its product to supply meaningful and timely management information, together with the use of those techniques to improve the process and its product*" [Goodman93].

Costello and Liu defined metric as "*A measurement derived from a software product, process, or resources. Its purpose is to provide a quantitative assessment of the extent to which the product, process, or resource processes certain attributes."*

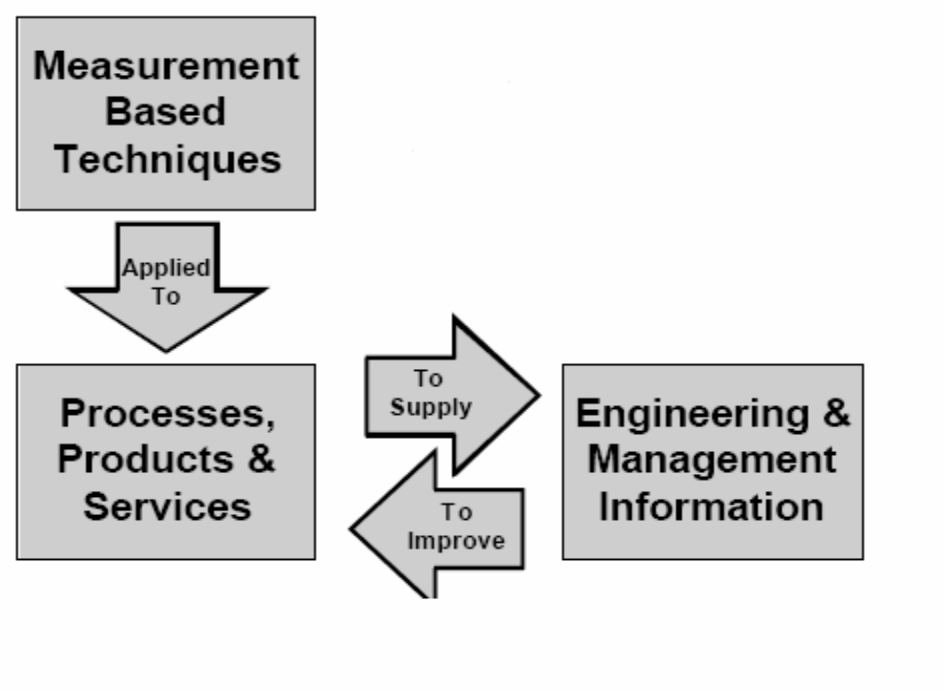

Fig 2: Information flow between engineers and managers

Fig 2 illustrates the information the flow between to engineers and managers, necessary to make good decisions at the right time.

3.3 Measurement

Measurement measure or calculates something. We use measurement in daily life. In the morning usually we drink a cup of coffee. Here the cup is a measure of the amount of coffee. Measurements are used almost in all situations in our lives. For example the distance between our room and the university, prices of ticket so on. Fenton defines measurement as:
"Measurement is a process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly define rules" [Fenton97].
Measurement contains information about attributes and entities. An entity is an object of real world. Example: person, table, pen, paper, cup. An attribute is a property or behavior of an entity. In the cricket game example, see twelth attribute of chapter 2, players are objects (entity) and their like bating and bowling are the attributes.

We are not only using measurements in our daily lives, but also in software engineering. Quality of our lives depends on software engineering, e.g., banking, airtrafic, satellites, and chemical plant and so on. Calculating or taking measurements in software

engineering is no luxury. According to Gilb, *Project without clear goals will not achieve their goals clearly.* In real time scenario, the project will be divided into modules, and for each module we can set goals and objectives to achieve them easily and efficiently.

3.4 Goal /Question/Metric (GQM) Technique

Metrics are usually just one aspect of an overall improvement program. The management collects the data and applies them to improve the process. Usage of metrics in the development will increase the quality of the production, reduce time and maintenance cost, increase the reusability and all this help to produce a product which is customer satisfactory. In this chapter we describe the basic steps of the Goal/Question/Metric (GQM). Later on (in chapter 5) we will apply the GQM to define metrics which can help us to improve the requirements management's process.

GQM [Basili and Rombach1988; Basili et al 1994] is a framework for developing a metrics program. A reliable metrics provides the evidence of improvements, allows cost-benefit analysis, and provides the basis for decision making. The GQM can be used in any phase of the software development. Measurement is a necessary prerequisite for software process improvement [Costello-Liu-95]. The reason to measure the software process, products, and resources is to characterize, evaluate, predict and improve the development process. Measures are like sensors, they tell to the developing team the project status, so as to produce an effective end product.

GQM is a technique that has been successfully used in the field of software engineering at NASA-SEL, Robert Bosch GmbH, Allianz Lebensversicherungs AG, Digital SPA Schlumberger RPS (CEMP Consortium, 1996) [Nick99]. The GQM paradigm consists of three steps:

- Generate a set of goals based upon the needs of the organization.
- Derive a set of question.
- Develop a set of metrics which provide the information needed to answer the question.

Generate a set of goals based upon the needs of the organization: This is the first stage of GQM here goals are set by the organization, so as to achieve a quality end product. The developing team sets goals for the improvement of an activity and is prioritized according to the organization strategy of development. Goals are defined in terms of purpose, perspective and environment [Linda95].

*Purpose***:** To (characterize, evaluate, predict, motivate) the (process, product, model, metric) in order to (understand, assess, manage, engineer, learn, improve) it.
Perspective: Examines the (cost, effectiveness, defects, changes, product metrics, reliability etc.) from the point of view of the (developer, manager, customer, corporate perspective, etc.).
Environment: The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints, etc.

Derive a set of Questions: The purpose of the questions is to understand and achieve the goals set by the organization. Questions tell the organization what to achieve in order to develop a quality end product. By using the question, we will not be deviated from the aim or objective of the goal. Examples of questions for the goal of improving the requirements gathering phase

Q1. What data is to be collected?
Q2. Does the collected data clearly depict what the organization has to build?
Q3. What is the quality of the document?

Questions are developed according to the goals set. The questions are like a guide to the requirements manager so that he will not be misguided or confused about the actions he has to do.

Develop a set of metrics that provide the information needed to answer the questions: In this step the actual data is identified. In the development process the data must be gathered evaluated, and validated. The identified data is evaluated to answer the questions to achieve the goals. Validation is done to check whether the entities given by the customer are achieving the goals or not. By doing this the developer can easily understand what are the entities and attributes and related measure to the project and develop a metric which is effective and efficient to attain maximum quality in the end product.

The goals set reflect the organizational needs to effectively accomplish a project. Relevant measures are derived on the basis of goals via a set of question to attain a quality end product.

3.5 Classification of Software Metrics

Grady [Grady92] classified software metrics into two types; primitive metrics and computed metrics. We can directly measure primitive metrics. Examples include Lines of Code (LOC) and number of errors in the program. No one has specified how to count LOC, whether to include commented line or only logical part and so on. Computed metrics, on the other hand are metrics computed using other metric values such as, quality of product, etc., that is these are not direct measurements.
According to Everald [Everald88] software metrics are classified into product and process metrics.

3.5.1 Product Metric

Product metrics measure the software as a product, from the initial stage to the last stage (product delivery). They also deal with the characteristics of the source code and measure the requirements, size of program, design and so on. If we define product metrics in the early stage of development, it will be a great help to control the process of the software development (product). Product metrics are size metrics, complexity metrics, Halstead's product metrics, and quality metrics.

Size Metrics

Size metrics measure the size of software. Measuring function points and bang system in the early stage of project gives more benefits.

1. Lines of Code (LOC): There is no rule to count the lines of code. The most common definition of LOC "*seems to count any line that is not a blank or comment line, regardless of the number of statements per line"* [Everald88]. We can easily predict the cost of product for modification or maintenance (using LOC we can predict the cost for module, using this we can calculate cost for maintenance).

2. Function Points: Albrecht proposed to measure the software size in the early stage of the product with function point *"The approach is to compute the total function points(FP) values for the project, based upon the number of external user inputs, inquires, output, and master files"* [Everald88]. Using FP we can measure the project size and effort required for development (using this we can measure cost by module).

3. Bang: DeMarco defines system Bang as a function metric, indicative of the size of the system. It measures the total functionality of the software system delivered to the user. Usually we calculate bang from certain algorithm and data primitives available from a set of formal specification for the software. Bang measures the functionality delivered to the user. *"A reasonable project goal is to maximize Bang per Buck- Bang divided by the total project cost"* [Everald88].

Complexity Metrics

Using Complexity Metrics we can manage or control the process of the software development and measure the complexity of a program.

1. Cyclomatic Complexity: Given any computer program, we can draw its control flow graph, G, where each node corresponds to a block of sequential code and each arc corresponds to a branch or decision point in the program. Using graph theory we can find out the complexity of the program, V(G) = E-N+2, where E is the number of edges and N is the number of nodes in the graph. For a structured program, we can directly find out the V(G), is the number of decision points in the program text [Everald88]. McCabe's cyclomatic complexity metric is related to programming effort, debugging performance and maintenance effort

2. Extensions to V(G): Myers noted that McCabe's cyclomatic complexity measure V(G), provides a measure of program complexity but fails to differentiate the complexity of some rather simple cases involving single conditions in conditional statements. Myers formula V(G) = [l:u], where l and u are the lower and upper bounds. Myers formula gives

more satisfactory results [Everald88]. Stetter proposed another formula to find out the complexity of the program. A function f(H) can be computed as a measure of the flow complexity of program H. The deficiencies noted by Myres are eliminated by f(H) [Everald88].

3. Knots: A knot is defined as a necessary crossing of directional lines in the graph. In a control flow graph, usually we take the nodes and block of sequential statements as knots. The number of knots in a program has been proposed as a measure of program complexity [Everald88].

4. Information Flow: The information flow of the program can also be used as metric for program complexity. Kafura and Henry as proposed the following measure:

$$C = [Procedure\ length] * ([fan\text{-}in - fan\text{-}out] * [fan\text{-}in - fan\text{-}out]).$$

Where, fan-in is number of local information flows entering, fan-out is number of exiting local information.

Halstead's Product Metrics

He proposed a set of metrics that will be applicable to many software products, and discussed the program vocabulary (n), length (N), and also total effort (E), and development time (T) for the software product.

Halstead defined program vocabulary N = N1 + N2, where N1 is the number of unique operators in the program and N2 is the number of unique operands in the program.
Program Length N=N1+N2, where N1 is the total number of operators in the program and N2 is the total number of operands in the program.
program volume V= N.log N, here N is the pure number.

Quality Metrics

Software Quality depends on correctness, efficiency, portability, maintainability, reliability, robust, etc. Usually we measure the software quality in every phase of the development cycle.

1. Detect Metric: The number of defects in the software product should be readily derivable from the product itself, it qualifies as a product metric [Everald88]. Everald proposed some measure to count the defect in the product,

- Number of design changes
- Number of errors detected by code inspections
- Number of errors detected in program tests
- Number of code changes required

Using these measures (values) we can predict the quality of the product

2. Reliability Metrics: Reliability metrics allow us to know the probability of software failures or rate at which software errors will occur. Using other metrics we can calculate mean time to failure (MTTF).

3. Maintainability Metrics: A number of efforts have been made to define metrics that can be used to measure or predict the maintainability of the software product [Everald88].

3.5.2 Process Metrics

Process metrics measure attributes of the software development process, like time, number of persons are involved, and types of methodologies. Here the interest is focused on the metrics that are best suitable to build a model that predicts plan and control software development. The metrics are used to improve the process and the management of the requirements that is carried out in the software development so as to produce an output which has maximum quality and attains customer satisfaction.

3.6 Scope of Software Metrics

Cost and effort estimation, productivity measures and models, data collection, quality models and measures, reliability, performance evaluation and models, structural and complexity metrics, capability-maturity assessment, management by metrics, evaluation of methods and tools, in all these areas we can use software metrics.

Chapter 4

Requirement Product Metrics

4.1 Introduction

High Quality Requirement documents are a must for successful software projects. Requirements are the initial step in every project, and therefore it is necessary to collect good requirement as early as possible in the project life cycle. All process phases directly and indirectly depend on the requirement document. S. Robertson and J. Robertson say

"I often say that when you can measure what you are speaking about, and express it in 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".

Using measures, we can understand which requirements are essential and using these measurements we can also set the goals. Using Requirement Engineering we can decide whether a requirement document is good or bad. The statement below tells us how well formed requirement documents should be.

"*A statement of system functionality that can be validated, that must be met or possessed by a system to solve a customer problem or to achieve a customer objective and that is qualified by measurable condition and bounded by constraints"*[AMOO].

Metrics should be useful and meaningful, only then can everyone understand the attributes (values) kept in the metric. In real time projects errors occur in the SRS document. Some authors (Davis and others) classify SRS errors into two types, Knowledge Errors and Specification Errors.

Knowledge Errors: These errors occur when people do not know what the exact requirements are. We can minimize these errors by doing prototyping. Remaining errors are rectified in the product development phase.

Specification Errors: These errors occur when people do not know how to adequately specify requirements. We have to remove these errors when we are preparing or writing an SRS document.

In chapter two we specified some important quality attributes of SRS. Here we describe some measure of these attributes.

4.2 Metrics

In this paragraph we propose a set of measures for the quality attributes described in chapter two. The attribute and measures are obtained my merging two tables described [IESE03] and are shown in Table 2.

$$\text{SRS Quality Equation } Q = \frac{\sum_{i=1}^{18} Qi\ Wi}{\sum_{i=1}^{18} Wi} \qquad \text{where, W is weight}$$

Using table 2 we can measure the SRS attributes.

Attributes	Formula	Purpose	Reference Value
Unambiguous	$Q_1 = \frac{N_{ui}}{N_r}$	To obtain the percentage of requirements that have been interpreted in a unique manner by all its reviewers	Close to zero, Ambiguous requirements. Close to one, Unambiguous requirements.
Correct	$Q_2 = \frac{N_c}{N_r}$	We calculate the percentage of all requirements that are valid.	0 = Incorrect. 1 = Correct.
Complete	$Q_3 = \frac{N_u}{N_i * N_s}$	To count the number of functions currently specified.	Grows closer to one, the more complete it becomes.
Verifiable	$Q_4 = \frac{N_r}{N_r + \sum C(R_i) + \sum T(R_i)}$	To measure the verifiability of a requirement.	0 = Very poor. 1 = Very good.
Understandable	$Q_5 = \frac{N_{ur}}{N_r}$	To count the number of requirements those are understood by all users (reviewers).	0 = No requirement understood. 1 = All requirements understood.
Modifiable	Not defined.	We can measure, if a table contains a context and index.	1 = If the table of context and index. 0 = otherwise.
Traced	Not defined.	Measuring the level of trace-ness is impossible, therefore no metric is presented	Not defined

Attributes	Formula	Purpose	Reference Value
Traceable	Not defined.	We can maintain a cross table for requirement and attributes, level wise.	1 = Traced attribute 0 = Otherwise.
Design Independent	$Q_9 = \frac{D(R_e \cup R_i)}{D(R_e)}$	To measure the percentage of possible solution systems that are eliminated by adding the overly constraining requirements.	1 = Design independent 0 = Highly design independent.
Annotated	Not defined.	To measure the percentage of requirements are annotated.	1 = SRS is not annotated. 0 = SRS is annotated.
Concise	$Q_{11} = \frac{1}{Size + 1}$	If two SRS documents are equal, we can choose best one using this formula.	1 = Best SRS. 0 = Worst SRS.
Reusable	Not defined	To determine if SRS is reusable or not.	1 = SRS is reusable. 0 = Not reusable.
Not Redundant	$Q_{13} = \frac{N_f}{N_u}$	Count the unique functions, which are not repeated.	1 = No redundancy. 0 = Complete redundant.
Achievable	Not defined.	The measure of the existence of a single system.	1 = A set of requirements are either achievable. 0 = Given acceptable development resources the SRS is achievable.
At Right Level of Detail	Not defined.	Is not possible to measure.	Not defined
Organized	Not defined.	Not defined	Not defined
Electronically Stored	Not defined.	If SRS is electronically stored, we can easily transfer the SRS.	1 = Electronically stored. 0 = Otherwise.
Consistent	Not defined.	The attribute is consistent.	Not defined.
Necessary	Not defined.	Not defined.	Not defined.

Table 2: Measures for attribute of SRS

Where,

- N_{ui} is the number of requirements for which all reviewers presented identical interpretations.
- N_r is the total number of requirements.
- N_i is the stimulus input of the function.

- N_s is the state input of the function.
- N_c is the number of correct requirements.
- $C(R_i)$ is cost necessary to verify presence of requirement R_i
- $T(R_i)$ is the time necessary to verify presence of requirement R_i
- Size is the number of pages.
- Re is the total of requirement that describe pure external behavior.
- R_i is the total of requirements that directly address architecture or algorithms of the solution.
- N_f is the total of functions currently specified.
- N_u is the current unique functions specified.

4.3 NASA Categories

Nine categories are developed from the NASA requirement documents selected from the Software Assurance Technology Center (SATC) library. These categories are used to indicate the quality of requirement document. The categories are divided into two classes; those are individual specification statements and entire or total requirement documents. I have taken all these categories from [SATC97]. These categories are used in the Automated Requirement Measurement tool.

4.3.1 Individual Specifications

1. Imperative: *Imperatives are those words and phrases that command that something must be provided.* Using Automated Requirement Measurement (ARM) tool, NASA has created a list of words/sentences classified as imperatives:

Shall is used to dictate the provision of a functional capability.

Must or **Must not** is used to establish performance requirements or constraints.

Is required to, is used as an imperative in specifications statements written in the passive voice.

Are applicable is used to include, by reference, standards or other documentation as addition to the requirements being specified.

Responsible is used as an imperative in requirements documents that are written for systems whose architectures are predefined.

Will is used to cite things that the operational or development environments are to provide to the capabilities being specified.

Should is not frequently used as an imperative in requirement specification statements. If it is used, the specifications statement is always found to be very weak.

2. Continuances: Continuance is used as an indication that requirements are organized and structured. Using this, we can indicate the presence of very complex and detailed requirement specification statements. Using ARM tool, NASA has specified some continuances, those are listed below.

3. Directives: *Directives is the category of words and phrases that point to illustrations within the requirement document.* They indicate of how precisely requirements are specified. Using the ARM tool, NASA has specified some directives, those are listed below.

4. Options: *Options is the category of words that give the developer latitude in satisfying the specification statements that contain them*. Using the ARM tool, NASA has found some options, those are listed below.

5. Weak Phrases: *Weak Phrases is the category of clauses that are apt to cause uncertainty and leave room for multiple interpretations.* Using the ARM tool, NASA has found some weak phrases, listed in Table 3.

Imperative	Continuances	Directives	Options	Weak Phrases
Shall	Below	Figure	Can	Adequate
Must or Must not	As follows	Table	May	As a minimum
Is required to	Following	For example	Optionally	As applicable
Are applicable	Listed	Note		Easy
Responsible	In particular			As appropriate
Will	Support			Be able to
Should				Be capable
				But not limited to
				Capability of
				Capability to
				Effective
				If practical
				Normal
				Provide for
				Timely

Table 3: Categories or Quality indicators

4.3.2 Entire requirement document

1. Size: *Size is the category used by the ARM tool to report three indicators of the size of the requirements specifications document, those are:*

Lines of text
Imperatives
Subjects of specification statements
Paragraphs

2. Text Structure: *Text Structure is a category used by the ARM tool to report the number of statements identifiers found at each hierarchical level of the requirements document.* Using these measurements we can indicate the consistency and level of details of the requirement document.

3. Specification Depth: *Specification Depth is a category used by the ARM tool to report the number of imperatives found at each of the documents levels of the text structure.*

4. Readability Statistics: i*s used to measure, how easily an adult can read and understand the requirement document*. Microsoft Word has proposed four types of readability statistics; Flesch Reading Ease index, Flesch-Kincaid grade Level index, Coleman-Liau Grade Level index, and Bormuth Grade level index.

Categories / Attribute	Impera-tives	Contin-uances	Dire-ctives	Options	Weak Phrases	Size	Text Structure	Spec. Depth	Readabil-ity
Unambiguous	×	×	×	×	×	×			×
Correct			×		×				
Complete	×	×	×	×	×	×	×	×	
Verifiable	×	×	×		×	×	×	×	×
Understandable	×	×	×	×	×	×	×	×	×
Modifiable	×	×					×	×	×
Traced									
Traceable	×	×					×	×	×
Consistent							×	×	
Annotated	×	×	×						×
Concise						×	×	×	×
Reusable			×				×		
Not Redundant			×				×	×	
Achievable	×	×	×	×			×	×	×
At Right Level of Detail	×	×	×				×	×	×
Organized									×
Electronically Stored									×
Necessary	×	×	×		×	×	×	×	×
Design independent			×						×

Table 4: Categories and Attribute

4.4 Automated Requirement Measurement Tool

The Automated Requirement Measurement (ARM) tool was developed by the Software Assurance Technology Center (SATC) at the NASA Goddard Space Flight Center as an early life cycle tool for assessing requirements that are specified in natural language. Understood by both parties (customer and developer), specifications are written in natural language. The use of natural language to prescribe complex, dynamic system has some problems, ambiguity, inaccuracy, inconsistency, etc [SATC97]. Using this tool we can find out the quality indicators (categories) like imperatives, weak phrases in requirement document. It is an aid to writing the requirements right, not writing the right requirements [SATC97]. If we apply the ARM tool on requirement document, it searches the entire requirement document line by line, identifies the quality indicators in the requirement document, and generates a file. In that file we find three types of reports, summary report, detailed imperative report, and detailed weak phrase report.

4.4.1 ARM Analysis Process:

The ARM tool looks at statement numbers/identifiers to analyze the structural level (how deep in the document) of the text. If an imperative (shall, must, etc.) is present, the line of text is considered to be a specifications statement. The words before the imperatives are subjects. It also counts the unique subjects, searches for continuances (end, etc.) and weak phrases (capability to, at a minimum).

4.4.2 Document Generated by ARM Tool

By analyzing the report generated by the ARM tool. The report is divided into category wise, we can judge whether the requirement document is good or bad.
The application of the ARM tool to a Software Requirement Specification is described in paragraph 4.4.3.

4.4.3. Software Requirement Specification document:

1.1 CPU card will provide a serial debug interface
2.1 TBS
3.1 CPU card's non-volatile RAM shall have a 10 year shelf life when stored at room temperature.
4.1 CPU card's non-volatile RAM can provide TBD days of data storage.
5.1 CPU card will provide 16 Mbytes of DRAM (dynamic RAM)
6.1 CPU card will provide 12 Mbytes of program FLASH
7.1 CPU card's Base-R interface shall be an RJ-45 connector on the back panel.
8.1 CPU Base-R connector shall use industry standard pinout.
9.1 CPU RS-485 interface may be a 9 pin female d-sub connector located on the front panel.
10.1 CPR card's R-chip interface pinout shall be TBD.
11.1 APU will provide 128 Kbytes of surge in a socket.
12.1 SPU card shall support switch over to a back-up in a normal amount of time.

Summary Report: The ARM Tool generated a report which gives us information about how many times each quality indicators occurred in the requirement document. It generates a table for each quality indicator. By applying the ARM Tool to the SRS we obtain the following:

IMPERATIVE	OCCURRENCE
Shall	5
Must	0
Is required to	0
Are applicable	0
Are to	0
Responsible for	0
Will	4
Should	0
TOTAL	9

Imperative Report: It gives the exact location of the imperatives, which are found in the requirement document.

Week Phrases Report: It lists the location and specifications contain indicators that are considered to be phrases that weaken the specification.

In the Appendix you can find the entire ARM tool output. Using ARM tool we can find out the quality indicators which we can use to modify the SRS document.

Chapter 5

Requirements Management Metrics

5.1 Introduction

Requirements Management (RM) is a skill to manage the requirements and their flow from one phase to the next phase in a project development. RM is concerned with meeting the needs of the customer and helps to estimate the activities in advance, like the over all cost that is needed for the development, number of people that is required in the development phase, date of delivery of the project and so on.

A requirement is a raw data given by the customer; RM takes the raw data, processes it and produces useful information which is understood by both the customer and the developer. A project that meets requirements by definition is a success [NCOSE95]. Requirements are taken from the customer in the form of natural language so as to avoid ambiguity, to have accurate data and a full understanding of what the customer needs, how he wants the end product and how it should work. Requirements are alive and active throughout the life cycle [NCOSE 95].

RM is an organized process which transforms inputs to outputs [Wiley02]. It provides design team for the development of high-quality systems. RM helps in developing a project and in managing and maintaining the project in a steady manner, even when change requests occur. RM is the key that gives a complete picture of the project; what is the status of the requirements? Who is responsible for a phase? What is the possible outcome of each phase? How are they interrelated? For an effective end-product, the requirement manager should maintain all these things effectively and efficiently.

5.2 Why Do We Need Requirements Management

The need for RM emerged in the1980's, when most of the software produced either failed or lacked the customer satisfaction due to poor RM. Most organizations have an immaturity in the RM field. Therefore, RM has become a key factor in any software developing industry. The people who are responsible for management of requirements should be efficient and also experienced, because experience teaches lots of knowledge on how to manage the requirements. RM is a kind of interface between the customer and the developing team (i.e., technical staff). The requirements manager acquires the requirements from the customer using different techniques like interviewing and brainstorming, that is questioning him/her to get a clear idea about the customer needs. The requirements are written in the most formal language so as to avoid ambiguity and focus on the goals and how to achieve them for customer satisfaction.

There are many techniques applicable to the requirements management. There are standards given by IEEE, ANSI and ISO. Among these is IEEE/ANSI 830-1993 the best one for stating SRS, as it covers all important aspects for an organization to consider while developing a project.
The advantage of following a set of standard rule is that we can attain our objectives fast and easily; set goals for the organization to achieve quality products. In any project development there are four main aspects: **time, cost, efficiency, reliability** and they are all interrelated. For instance, if more **time** is taken while developing the project then more resources (**cost**) have to be spent on the developing team. If the developed project is **efficient** then the customer can **rely** on the product. Following standard rules can give an **efficient** product at a **reliable cost** at a minimal **time**.

Apart from standard rules, we can develop our own RM process. It is a tedious work and the person who has this task should have a very good knowledge on the requirements; that is, how to collect them from the customer without giving scope to ambiguity and inconsistency. To attain a quality output, the requirements manager should have experience in the field of requirements, as requirement management is the most critical part of any project development. So, experience counts very much. A good requirements manager will be able to produce a SRS which exhibits the following characteristics [Omni99]:

- Lack of ambiguity.
- Completeness.
- Consistency.
- Traces to origins.
- Absence of design.
- Enumerated requirements.

The following standards are better than developing your own RM process as they saves money and also the appreciable amount of time that is spent on acquiring the requirements.

5.3 Tasks of Requirement Management

There are nine major tasks in Requirement Management. The tasks are [NCOSE95]:

- Define and communicate what is wanted.
- Trace of requirements.
- Requirements applied on the solution.
- Requirements for optimizing the system.
- Requirements to drive design and implementation.
- Manage change, problem reports and suggestions.
- Manage the portioning of work to specialists.
- Testing and validating of the finished product.
- Requirements assist project development.

5.3.1 Define and communicate what is wanted

Defining requirements is the most important and first task of any software development. Once we have defined what is wanted then the rest follows easily for the development team. The RM should establish a good communication between the customer and the developing team. With a good communication the development team can be more focused on the goal. Most software's fails due to poor communication between the requirements manager-customer, customer-developer and even between the requirements manager-developer. Therefore communication plays an important role in defining the project goals.

5.3.2 Trace of requirements

Tracing the requirements give information about the status of every requirements i.e., whether the requirements are yet to be used, still in the processing or already finished. Tracing requirements to their more abstract predecessor requirements and the more detailed successor requirement is a vital aspect in development [Omni99]. All requirements are documented at one place instead of placing the requirements in multiple documents. This helps to track the requirements easily and quickly, and to access their status.

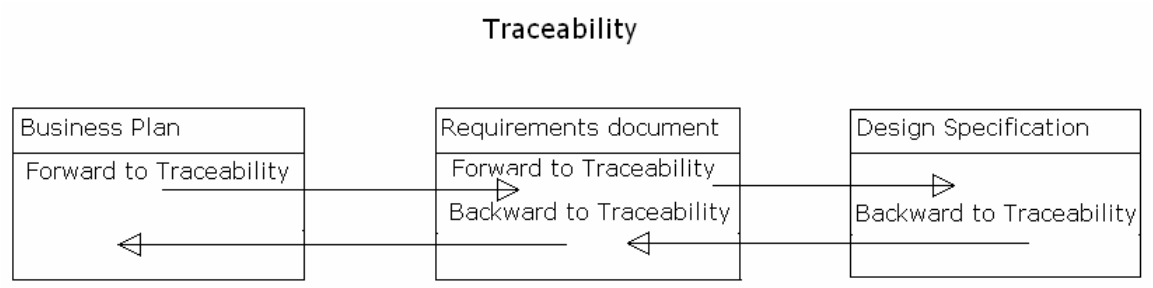

Fig.4: Trace of requirements

5.3.3 Requirements applied on the solution

After acquiring the formal requirements the requirements manager prepares an abstract model and applies the requirements to get a clear view of the end product. It's a trail and error method to choose which model is most appropriate for development.

5.3.4 Requirements for optimizing the system

The requirements are grouped and each group is given a priority on a cost-benefit basis. After prioritizing the requirements, they are used to develop a software system. Some are delayed, if they have a low priority or are not important at the beginning of the project. These can be implemented at last or left according to the user needs. Some requirements

are very important at the start of the development of the project and those are given high priority and must be done well to ensure customer satisfaction. This kind of optimizing gives a new shape to the design of a project.

5.3.5 Requirements to drive design and implementation

After defining the requirements and prioritizing the requirements into groups, we find a suitable model to develop the project. After selecting a model we start the design and in parallel tracing is done to check the status of the requirements.

5.3.6 Manage change, problem reports and suggestions

In a project the requirements are gathered not only in the first phase, while developing the project the developer can get new requirements. Managing new or change requirements is a vital issue in any project. A good set of requirements defines precisely what is wanted, but simultaneously leaves the maximum space for creative design [NCOSE95].

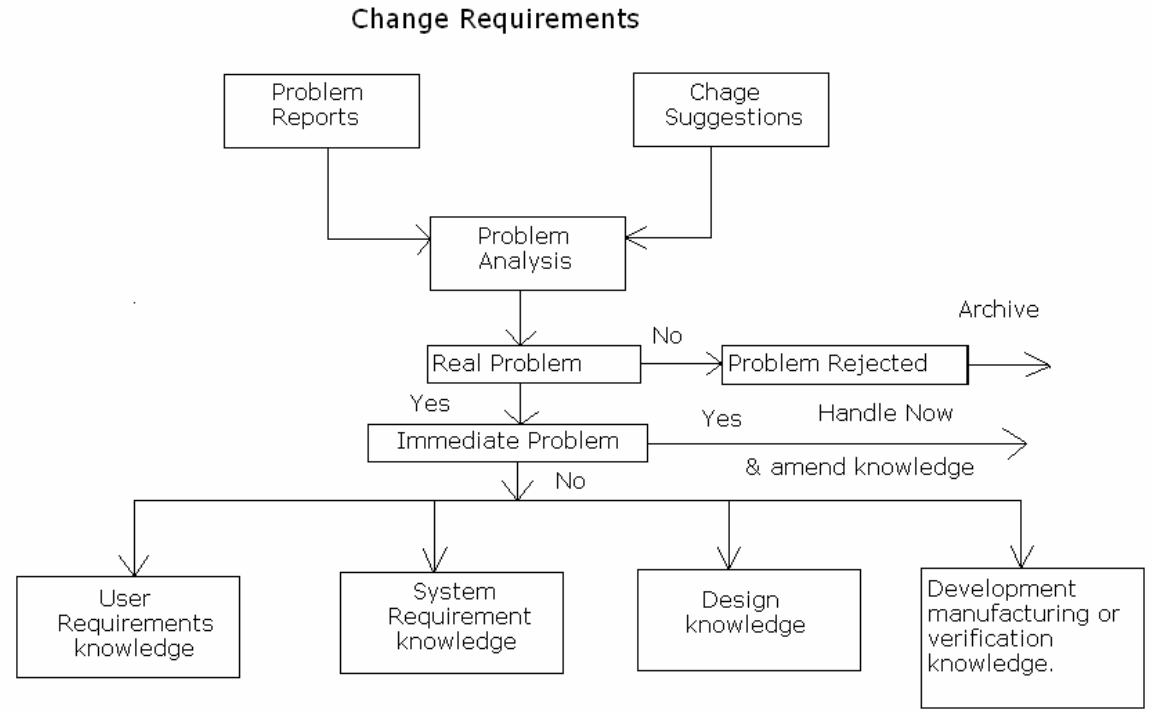

Fig.5: Change Requirements

Fig 5 shows an overview of what happens when there is a change in requirements while it is in the development of the project.

5.3.7 Requirements for portioning into sub-systems

Controlling a large process is very difficult. A major process is divided into modules (sub-process), and specialized people are assigned to these modules and requirements are

given them. Then the development is carried out and all the modules are combined into one final product at the end. Thus we can attain an effective end product. A major task in this process is to keep track of all the modules and grouping them into a final product.

5.3.8 Requirements for testing the product

Testing is another important phase in the development. After defining the requirements and the goals, they are transformed into logical programming code. While the code is developed, it is also tested in parallel. This is the most effective way to produce a quality product. Testing is done to remove bugs to assure a bug-free product.

5.3.9 Requirements to assist project management

Requirements management should effectively manage the project milestones, interfaces with external systems and attain the goals of the project. RM is not only concerned with the development but also useful in decision making. E.g., while selecting a model the RM should check the costs and time consumed. The RM should also take effective decision for both project and organization.

These are the tasks necessary to obtain the requirements effectively and efficiently from the customer. The requirements manager should perform these tasks to improve the acquirements of requirements from the customer and write them down properly for the better understanding for the developing team to develop the software process.

5.4 Models for Requirements Management Process

There are many models to do requirement management effectively. In general the Requirement Engineering Process (RM) is a process with inputs and outputs. The inputs and outputs are [Wiley02]:

Existing System Information (I): Before initializing a project the requirement manager has to complete a check list of the organizations resources. The requirements manager then matches them with the customer needs to check whether all the resources are acquired to develop the project.

Stakeholders Needs (I): These give the information about the customers needs. That is, how the customer is not concentrated with how the system is built, but the customer is concerned with the end product.

Organizational standards (I): These give the standard rules that are followed by the organization regarding the development and quality management.

Regulations (I): These concerned the data, i.e., whether the data is safe or not.

Domain Information (I): gives general information about the application information that concerns the system.

Agreed requirements (O): After analyzing the final requirements specification, it should be agreed by the customer and the developer to start the project.

System specification (O): gives a detailed description of the working model (prototype) of the developed system.

System Models (O): Describes the system from different perspectives i.e., how the development is going to be done.

Requirement Engineering Process: This is a key process which takes the inputs (I) and processes them to produce an output (O)

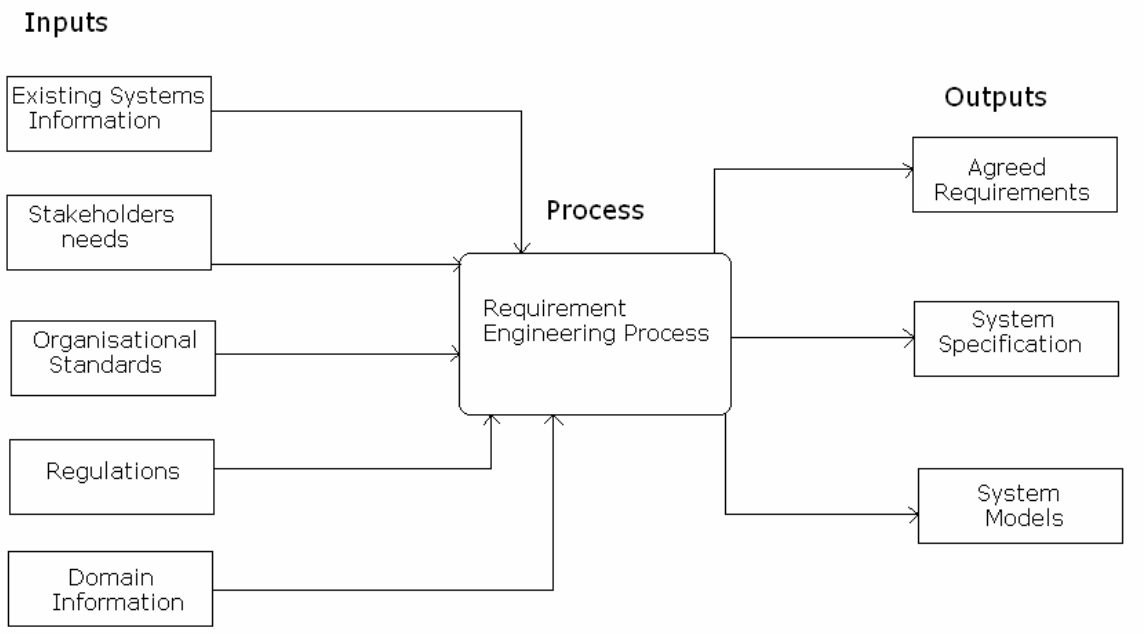

Fig.6: Input and Output of a Requirements Engineering Process [Wiley 02]

The model shown in Fig 6 gives an overview of the process. This provides general information about the process, and tells the requirement manager how to start and what the necessary resources are. While developing a project, the developer can use more than one model to attains the target. Kotonya and Sommerville have listed some models for requirements management [Wiley02]. They are listed in the coming paragraphs…

5.4.1. Coarse-Grain activity models: This is the most primitive RM process model. Much software fails due to poor RM. Therefore the need for RM has increased in software development area. It was recognized in late 80's that the use of RM has increases the quality and efficiency of software. The pictorial diagram for Coarse-Grain model is as shown in Fig 7.

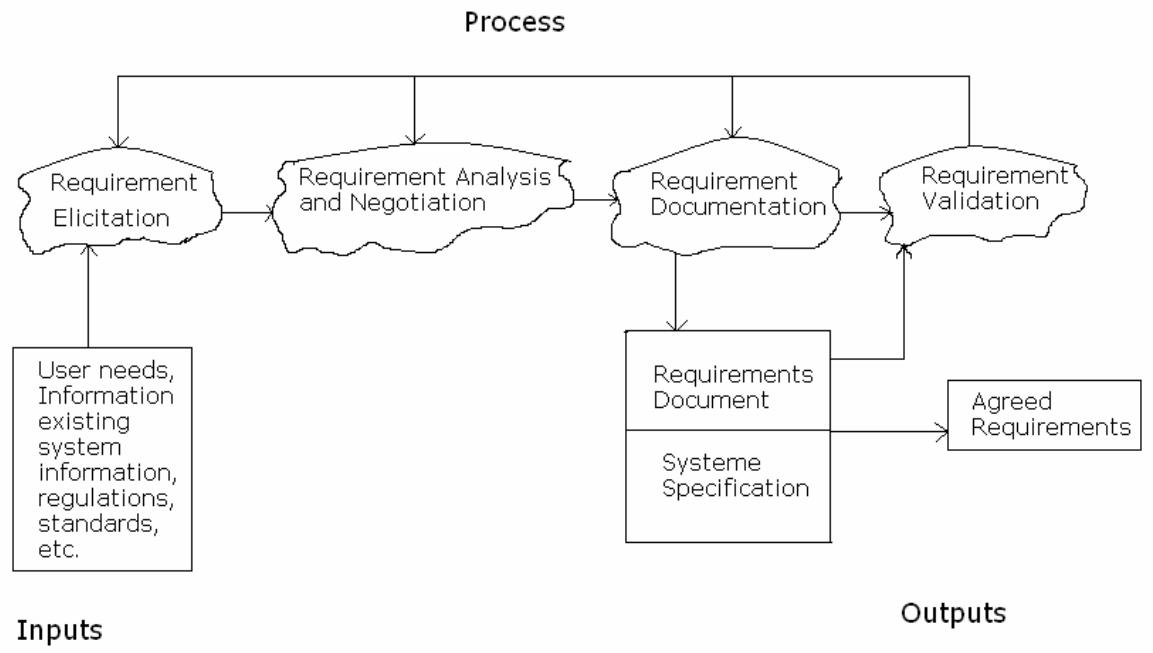

Fig.7: Coarse-Grain Activity Models

The inputs in Fig 7 are user needs, information about the existing system regulations and standards. Using these inputs as resources the Coarse-grain models process the requirements. The phases of this model are:

a) Requirements Elicitation: Requirements are discovered with the consultation of the stake holders. It is also called as requirement acquisition.

b) Requirements Analysis and Negotiation: The requirements are analyzed in detail and then negotiated with the stakeholders, to determine whether or not they are satisfactory to them or not.

c) Requirements Document: The analyzed and negotiated requirements are documented for further usage.

d) Requirements Validation: In this phase the requirements are carefully validated i.e., checked for consistency and completeness.

The Coarse-grain model gives information about the requirements elicitation, analysis and negotiation, documentation and validation. It does not reveal anything about how the process is developed or what attributes must be considered in a certain phase.

5.4.2. Fine-Grain Model: This method is a process that divides the project in to finer level of granularity and assigns people who are specialized in that area. The decomposition of a module is the initial step in this model, made to attain maximum understandability of the process see Fig 8. The process is performed by a single

individual or a group of participants in the development life cycle. This model is also concerned with combining the individual processes in order to achieve the overall objectives of the development. Good coordination between the individual processes can be acquired with good cooperation and communication [Fine-Grain].

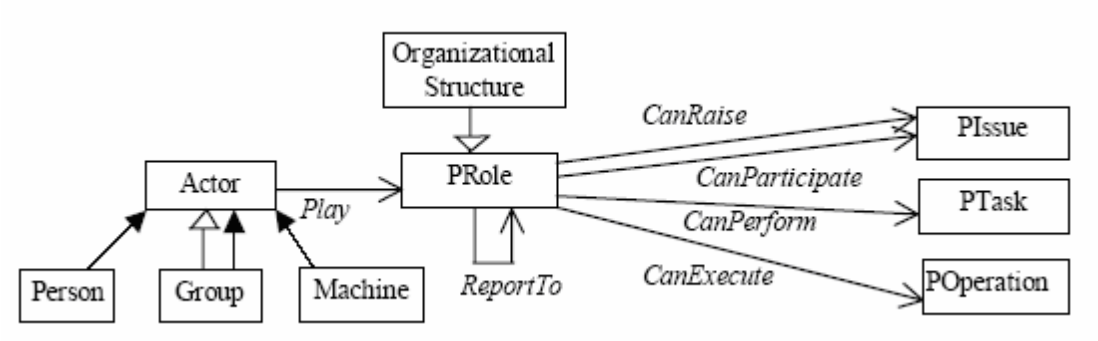

Fig.8: Fine-Grain Model an Organization View [Fine-Grain]

5.4.3. Role-Action Model: This model explains the different people involved in the process and their assigned activities in the development process. This process shown in Fig 9 is more helpful in process understanding and automation.

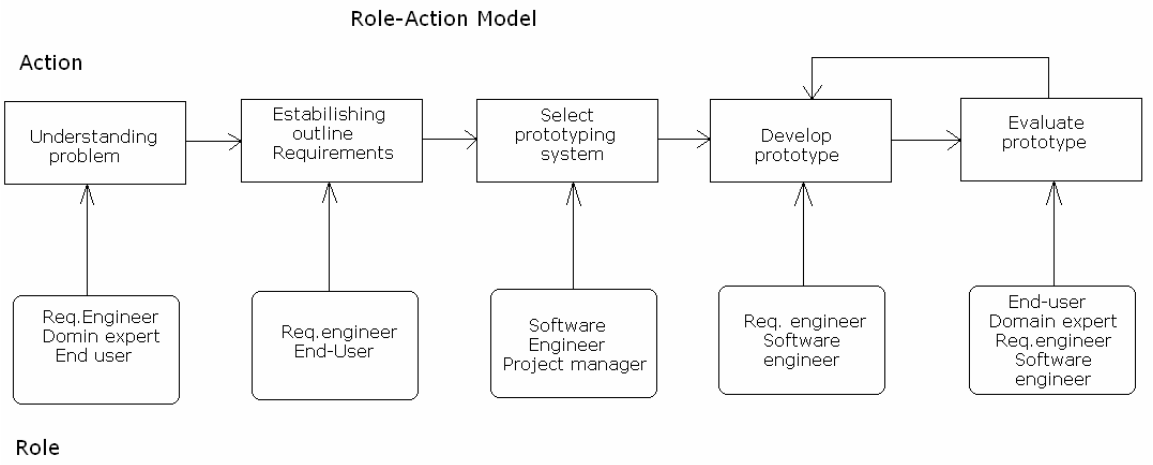

Fig.9: Role-Action Model

Table 5 summarizes the processes and that can be used to manage requirements and the application area for which they are more suitable

Process	Subjected Area
Coarse-Grain Model	Provides an overall picture of the process.
Fine-Grain Model	Used for better understanding and improving the processes.
Role-Action Model	People Management

Table 5: Process and their Subjected Area

In parallel with the above processes we also need to manage change to the requirements. Changing is inevitable as business priorities change, errors or omissions in the requirements are discovered and as new requirements emerge. The process for changing requirements is more effectively done in the software development life-cycles. In the Waterfall model which is more useful for computer based systems the development starts from requirements gathering and then follows software design process.

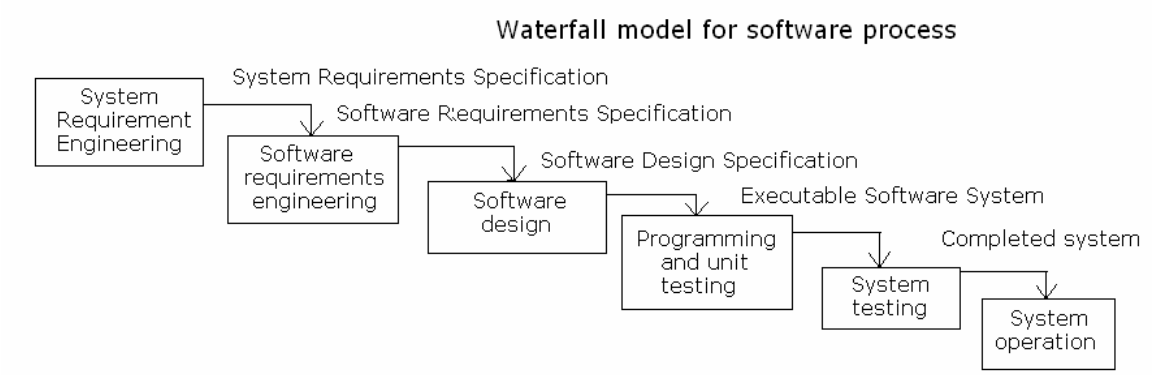

Fig.10: Waterfall Model for Software Process. [Wiley 02]

This model clearly depicts what the requirements are in each phase of the Waterfall development life cycle.

In a software development life cycle that uses **Spiral model**, the Requirements process in this model is as follows,

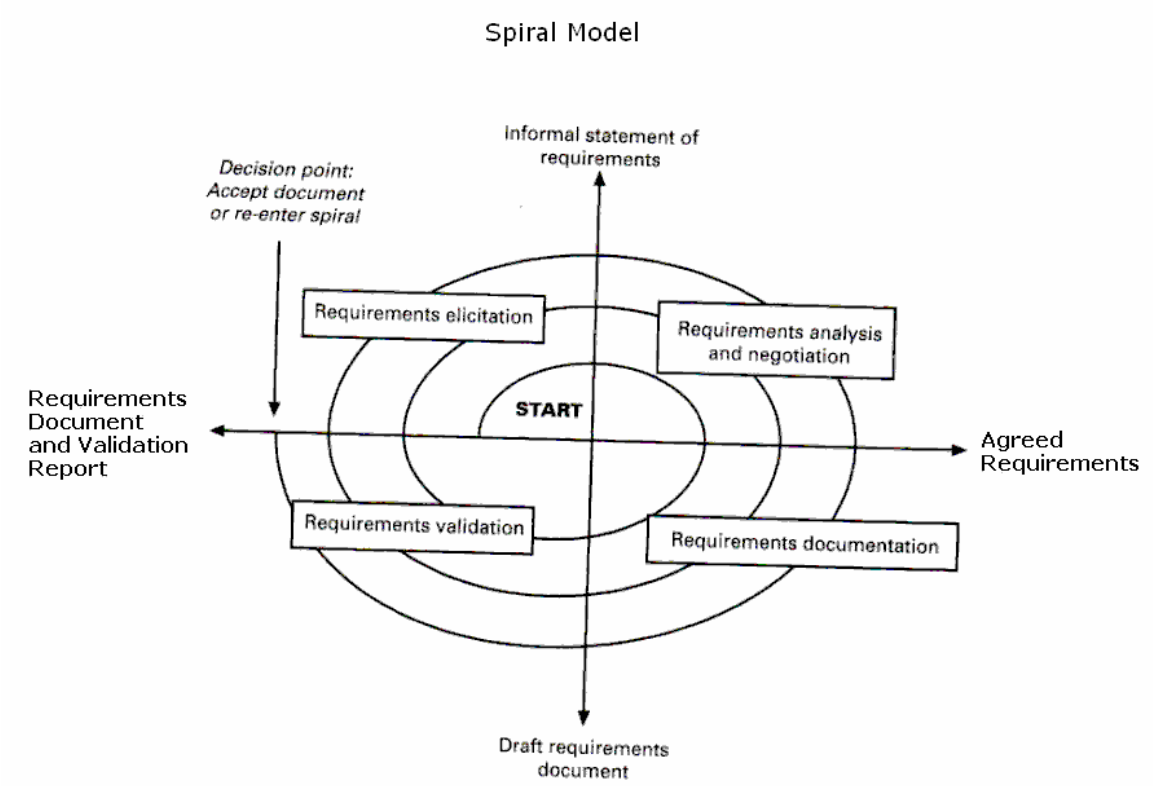

Fig.11: Spiral Model [Wiley 02]

The Requirements Manager does a detailed study together with the customer to gain clarity on the requirements of the customer and to choose a model from those listed above which is cost effective and reliable. The RM manager establishes a good interaction and coordinates the phases to achieve the overall objectives of the project or the organization.

5.5 Applying GQM Technique to Improve the Requirements Management Process

GQM helps in solving problems by developing goal driven measurement. Measurement is a necessary prerequisite for software process improvement [Costello-Liu95]. The reason to measure the software process, products, and resources is to characterize, evaluate, predict and improve the development process. Measures are like sensors, they tell to the developing team the project status, so as to produce an effective end product. The measurement provided by GQM model is to predict the relationship between the attributes to the process development which are used to establish achievable goals for cost, schedule and quality.

GQM has been proposed and applied as a systematic technique for developing a measurement program for software processes and products. GQM can be used in any phase of the development. The measurement provided by the GQM reflects more or less of Quality Improvement Paradigm (QIP). As GQM is one of the main components of QIP [GQM98].

The GQM is based on three main tasks that is identifying the goal, how to obtain the measurement which reflects on the metric which is used to improve the process. The measurement starts from identifying the goal, and questionnaire to build a metric to collect data related to achieve the goals [Linda-Lawrence97].

Identify the goals: Goals are set by the organization to improve the performance of any phase. The goals are defined in terms of purpose, perspective and environment. Goals are like the building blocks to create an estimation or plan for the development. Here, we referred some papers, and finally we decided these goals are most important.

G1: Improve the requirements gathering.
G2: Improve the requirements process.
G3: Improve SRS quality.
G4: Improve software estimation.
G5: Minimize schedule.
G6: Minimize development cost.
G7: Improve software quality.
G8: Improve productivity.

These eight goals constitute the improvement of the software development process. In this the first three goals mainly concentrates on requirements and the rest five can be used in other phases of development.

Identify the questions: Questions are used to define what data is to be collected. Questions are derived from the goals set by the organization in the first step. From these formulated questions answers are found to achieve the goals. We developed twenty-nine questions for each goal by consulting [SPC] [Linda95]. The questions are shown in the following paras.

Identify the metric: A metric is used to achieve the goal set by the organization. In this step we develop the metric to achieve the goals and questions for improving a phase. After this step we collect the data, which are used to achieve the goals and questions set by the organization [SPC] [Linda95].

Using above steps we developed metrics to improve the requirements phase.

G1: Improve the requirements gathering.
The aim of this goal is to improve the requirements gathering e.g., assign more experienced and knowledge people to gather requirements. To achieve this goal we developed four questions and metrics (see the Table 6).

Q1- What are the customer needs?
Here the customer tells the requirements manager his needs, like what are the functions he needs in the system. It is just like abilities of the system.
Q2- What are the assigned tools?
Assigned tools for the development are achievable or not.
Q3- What are the requirements provided by the customer for the development?
Requirements provided by the customer for the development.
Q4- What are the change in requirements?
Requirements are not only gathered in the beginning of the development, they also change during the development. So, the requirements must be constantly tracked, if any new requirements are added they should be managed efficiently.

Questions	Metrics	Measures
Q1	Collect Requirements	• Number of Requirements
Q2	Available tools for requirements	• Number of tools
Q3	Functional Requirements	• Number of Functional Requirements
Q3	Non-Functional Requirements	• Number of Non-Functional requirements
Q4	Change Requirements	• Number of changed Requirements

Table 6: Metrics for G1

G2: Improve the requirements process.
The aim of this goal is to improve the requirements process e.g., involving technical staff of the project in this phase to get a clear idea of requirements. To achieve this goal we developed two questions and metrics (see Table 7).
Q1- How many people are involved in the requirements phase?
Total number of people working in a phase like requirements gathering people, programmers, developers, testers, customers, users, requirements managers and project managers.
Q2- What are the abilities of programmers, designers, testers are required?
Working abilities of the people assigned for the development.

Questions	Metrics	Measures
Q1	Number of people involved in the Requirements phase	• Number of programmers • Number of designers • Number of testers • Number of customers • Number of users • Number of requirement engineers • Number of project managers
Q2	Change Requirements	• No of Requirements change

Table 7: Metrics for G2

G3: Improve SRS quality.
After writing the SRS document verify it with ARM tool to find out the quality indicators in the SRS document and modify the SRS document accordingly. We developed three questions and metrics (see Table 8).

Q1- What are the attributes considered?
The attributes considered for better understanding of the SRS document.
Q2- How many errors are found in the SRS?
Errors found while evaluating the SRS. The measure is the number of errors.
Q3- How many are solved, deleted of those errors?
By evaluating the SRS once again and re-tracking the document.

Questions	Metrics	Measures
Q1	Attribute considered	• Number of attributes
Q2	Errors found in SRS	• Number of errors found
Q3	Errors corrected in SRS	• Number of corrected
Q3	Errors deleted in SRS	• Number of errors deleted
Q3	Modified errors in SRS	• Number of errors modified

Table 8: Metrics for G3

G4: Improve software estimation.
Here we estimate the cost, size and efforts of the development team to avoid maximum risk in the development. To achieve this goal we developed three questions and metrics (see Table 9).

Q1- What is actual vs. estimated labor charges for an activity?

Cost of the labor spent on each activity, is measured in PH/LOC (person hour per line of code)
Q2- What is the actual vs. estimated schedule, effort, and size for each activity?
To verify the degree of accuracy of all the activities in the development.
Q3- What is the actual vs. estimated staffing level? Overtime worked?
It is the possible date of delivering the project in time. Track the overtime spent on the project delivery in cost.
Q4- How many requirements are changed for each activity?
Here we list the number of changed requirements for each activity.

Questions	Metrics	Measures
Q1	Initial estimate vs. actual effort for each activity	• Estimates number of person hours to complete • Actual number of person hours to complete
Q2	Initial estimate vs. actual project schedule for each activity	• Estimated start date and completion date for each activity
Q2	Initial estimate vs. actual size of the software	• Date activity started and completed • Estimate SLOC of new code
Q3	Total overtime hours	• Estimate SLOC of resued code
Q3	Labor rate for each activity	• Number of overtime worked
Q4	Requirements changed for each activity	• Number of persons per each activity • Number of Requirements deleted • Number of Requirements added • Number of Requirements changed

Table 9: Metrics for G4

G5: Minimize schedule.
Creating a schedule for a project is a tuff task. Here we allocate time for each activity for the completion of the project within the specified time. To achieve this goal we specified three questions and metrics (see Table 10).

Q1-What is the schedule for each activity?
To minimize the schedule one should know exactly how much time is spent on the activity. Measure actual schedule for each activity.
Q2-What is the actual level of effort for each activity?
Effort spent on each activity to understand the actual schedule for the activity.
Q3- How much time spent on other activities?

Time spent on fixing the errors, non-development tasks like support, maintenance, and administrative tasks.

Questions	Metrics	Measures
Q1	Initial estimate vs. actual effort for each activity.	• Project start date • Estimate completion date for each activity • Actual completion date for each activity
Q2	Person hours spent on rework.	• Estimate number of persons hours to complete • Actual number of persons hours to complete
Q3	Total hours spent on other activities.	• Total number of hours required to fix defects • Number of overtime hours worked

Table 10: Metrics for G5

G6: Minimize the development cost.
Here we estimate the amount spent on the total project but also for each activity in the development. This helps to minimize the overall cost of the project. To achieve this goal we specified six questions and metrics (see Table 11).

Q1- What is the cost for each activity?
We compare the estimated vs. actual cost spent on each activity.
Q2- What is the labor rate for each activity?
Labor rate is measured in PH/LOC (person hour per lines of code). Accurate value for labor rates is essential.
Q3- What is the estimated vs. the actual cost for each activity?
Here we estimate the cost for each activity.
Q4- How much of the budget is spent on development vs. managerial vs. support tasks?
Breaking the budget down by various categories of employees may indicate areas where cost could be saved.
Q5- Amount spent on the errors correction?
Amount spent on finding the errors, detecting them and correcting them.
Q6- How much money spent on maintenance?
Money spent on maintaining the software after development.

Questions	Metrics	Measures
Q1	Actual cost for each activity	• Total cost • Initial cost estimate • Funds allocated for each activity or phase
Q2	Labour rate for each activity	• Total SLOC produced • Person hours to complete.
Q3	Initial estimate vs. actual effort for each activity	• Estimated number of person hours to complete • Actual number of person hours to complete
Q4,Q6	Percent of budget spent on management task	• Total amount spent on management tasks
	Percent of budget spent on support task	• Total amount spent on support tasks
Q5	Amount spent fixing defects in each activity	• Persons hours spent on fixing the defects

Table 11: Metrics for G6.

G7: Improve software quality.

To assess the reliability and maintainability of the product, we measure the software quality as the average number of defects per size unit of code [SPC]. We specified three questions and the corresponding metrics (see Table 12).

Q1- How many defects are there in the product?
We determine how many defects were detected, corrected, and modified.
Q2- Is the software reliable, maintainable?
We try to assess the degree of difficulty involved in updating the software.
Q3- Is the software stable?
Determine the stability of the system by measuring the frequency with which the software becomes inoperable.

Questions	Metrics	Measures
Q1	Average person hours to fix a defect Meantime between failure Number of defects detected in each phase Number of defects/SLOC	• Average PH to fix a defect • Meantime between failures • Total number of defects detected • Total SLOC produced
Q2	Total Lines of Document (LOD)	• Total LOD
Q3	Percent of code inspected	• Number of units coded (new) • Number of units inspected

Table 12: Metrics for G7

G8: Improve productivity.

Productivity is defined as amount of work that can be performed in a unit of time for instance in one day or one week. To achieve this goal we consider all possible metrics so as to improve productivity. We specified four questions and a corresponding metrics (see Table 13).

Q1- How much time is being spent on rework?

Estimating the amount of time spent on rework over a number of projects, we can determine a company average for rework time.

Q2- Are developers spending too much time on support and managerial activities?

If the project is understaffed in management and /or support position, the developers will need to perform these tasks to keep the project moving. The more this happens, the more overall productivity will decrease. By maintaining data on the type of work performed by development staff, you should be able to identify optimal management and support staff levels.

Q3- Is the productivity consistent with the experience of the team members?

By employing experienced people on the project, it will increase the performance of development to get a quality product.

Q4- Are tools available to designers and managers?

By using the efficient tools to the designers and managers we can improve the efficiency of the end product.

Questions	Metrics	Measures
Q1	SLOC/person hour for each activity	• Total SLOC produced • Actual number of person hours to complete
Q2	Percent of budget available for support staff	• Current total budget • Total money allocated to support staff
Q3	Ratio of development staff per manager Number of staff at each experience level	• Number of managers for each activity • Number of development staff for each activity • Number of staff at each experience level
Q4	Percent of budget available for software development tools	• Amount spent on development tools

Table 13: Metrics for G8

These questions and metrics are drawn to achieve the goals. The set of goals defined reflect the organizational needs. Relevant measures are derived on the basis of goals via a set of question. By using the above GQM technique we can improve the requirements phase, but also other phases of the development. By following the above mentioned steps the requirements manager can efficiently manage the development to produce a high quality output.

By using the GQM the requirements manager can build a good design, draw objectives, and attributes of the project which are reliable, cost effective, to attain maximum quality.

Conclusions

In this thesis we have proposed a set of product metrics to improve the quality of the software requirements specifications and a set of management metric to improve the requirements management activities.

Together with the product metrics, we proposed nineteen quality attributes of SRS. Using these attribute we can build an high quality SRS document. To create an high quality SRS we can also use the Automated Requirement Measurement Tool (ARM Tool) which analyze the text of a SRS with respect to some quality indicators (see chapter 3) and produces a report which suggest the quality level of the SRS document. An example of a report generated by the ARM Tool can be found in appendix.

The requirements management measures were obtained by applying the Goal/Question/Metric (GQM) method. We define eight goals, twenty-nine questions and thirty-seven metrics (see chapter 5).

The set of product and management metrics proposed in this thesis are not exhaustive, more attributes and metrics can be defined. However we feel confident that using these metrics we can improve the quality of the requirements and of the end software product.

Appendix

SRS Document:

1.1 CPU card will provide a serial debug interface
2.1 TBS
3.1 CPU card's non-volatile RAM shall have a 10 year shelf life when stored at room tempature.
4.1 CPU card's non-volatile RAM can provide TBD days of data storage.
5.1 CPU card will provide 16 Mbytes of DRAM (dynamic RAM)
6.1 CPU card will provide 12 Mbytes of program FLASH
7.1 CPU card's Base-R interface shall be an RJ-45 connector on the back panel.
8.1 CPU Base-R connector shall use industry standard pinout.
9.1 CPU RS-485 interface may be a 9 pin female d-sub connector located on the front panel.
10.1 CPR card's R-chip interface pinout shall be TBD.
11.1 APU will provide 128 Kbytes of surge in a socket.
12.1 SPU card shall support switch over to a back-up in a normal amount of time.

ARM Tool output:

```
                AUTOMATED REQUIREMENT MEASUREMENT (ARM)

                           SUMMARY REPORT

                   FOR SPECIFICATIONS CONTAINED IN

                          THE FILE Sample.txt

     ARM REPORT for file Sample.txt on 03-03-05, at 14:11, Page-1

IMPERATIVES are those words and phrases that command that something
must be provided.

     "SHALL" normally dictates the provision of a functional
     capability.

     "MUST" or "MUST NOT " normally establish performance
     requirements or constraints.

     "WILL" normally indicates that something will be provided from
     outside the capability being specified.

An explicit specification will have most of its counts high in the
IMPERATIVE list (i.e. shall, must, required).

The counts of IMPERATIVES found in file Sample.txt are shown
in the table below.


          IMPERATIVE                  OCCURRENCE
          --------------------        ----------
          shall                              5
          must                               0
          is required to                     0
          are applicable                     0
          are to                             0
          responsible for                    0
          will                               4
          should                             0
                                      ----------
                         TOTAL               9

     ARM REPORT for file Sample.txt on 03-03-05, at 14:11, Page-2


CONTINUANCES are phrases such as "the following:" that follow an
```

imperative and precede the definition of lower level requirement specifications. The extent that CONTINUANCES are used is an indication that requirements have been organized and structured. These characteristics contribute to the tractability and maintenance of the subject requirement specification document. However, extensive use of continuances indicate multiple, complex requirements that may not be adequately factored into development resource and schedule estimates.

The counts of CONTINUANCES found in file Sample.txt are shown in the table below.

CONTINUANCE	OCCURRENCE
below:	0
as follows:	0
following:	0
listed:	0
in particular:	0
support:	0
and	0
:	0
TOTAL	0

ARM REPORT for file Sample.txt on 03-03-05, at 14:11, Page-3

DIRECTIVES are words or phrases that indicate that the document contains examples or other illustrative information. DIRECTIVES point to information that makes the specified requirements more understandable. The implication is the higher the number of Total DIRECTIVES the more precisely the requirements are defined.

The counts of DIRECTIVES found in file Sample.txt are shown in the table below.

DIRECTIVE	OCCURRENCE
e.g.	0
i.e.	0
For example	0
Figure	0
Table	0
Note:	0
TOTAL	0

ARM REPORT for file Sample.txt on 03-03-05, at 14:11, Page-4

OPTIONS are those words that give the developer latitude in the implementation of the specification that contains them. This type of statement loosens the specification, reduces the acquirer's control over the final product, and establishes a basis for possible cost and schedule risks.

The counts of OPTIONS found in file Sample.txt are shown in the table below.

OPTION PHRASES	OCCURRENCE
can	1
may	1
Optionally	0
TOTAL	2

ARM REPORT for file Sample.txt on 03-03-05, at 14:11, Page-5

WEAK PHRASES are clauses that are apt to cause uncertainty and leave room for multiple interpretations. Use of phrases such as "adequate" and "as appropriate" indicate that what is required is either defined elsewhere or worst, the requirement is open to subjective interpretation. Phrases such as "but not limited to" and "as a minimum" provide the basis for expanding requirements that have been identified or adding future requirements. WEAK PHRASE total is indication of the extent that the specification is ambiguous and incomplete.

The counts of WEAK PHRASES found in file Sample.txt are shown in the table below.

WEAK PHRASE	OCCURRENCE
adequate	0
as appropriate	0
be able to	0
be capable of	0
capability of	0
capability to	0
effective	0
as required	0
normal	1
provide for	0
timely	0
easy to	0
TOTAL	1

ARM REPORT for file Sample.txt on 03-03-05, at 14:11, Page-6

INCOMPLETE is the category of words and phrases that indicate that the specification of requirements is not fully developed or provides a basis for expansion or addition of new requirements at a later date.

"TBD" indicates that necessary information has yet TO BE DETERMINED.
"TBS" indicates that a required event has yet TO BE SCHEDULED.
"TBE" indicates that a needed designation has yet TO BE ESTABLISHED or yet TO BE ESTIMATED.
"TBC" indicates that a needed value has yet TO BE COMPUTED.
"TBR" indicates that a question regarding a condition or value as yet TO BE RESOLVED.
"not defined" and "not determined" are phrases that explicitly declare that a specification statement is incomplete.
"but not limited to" and "as a minimum" are phrases that open the requirements specifications to future modifications or additions.

The counts of INCOMPLETE words and phrases found inSample.txt are shown in the table below.

INCOMPLETE	OCCURRENCE
TBD	2
TBS	1
TBE	0
TBC	0
TBR	0
not defined	0
not determined	0
but not limited to	0
as a minimum	0
TOTAL	3

ARM REPORT for file Sample.txt on 03-03-05, at 14:11, Page-7

NUMBERING STRUCTURE DEPTH provides a count of the numbered statements at each level of the source document. These counts provide an indication of the document's organization and consistency and level of detail. High level specifications will usually not have numbered statements below a structural depth of four. Detailed documents may have numbered statements down to a depth of nine. A document that is well organized and maintains a consistent level of detail will have a pyramidal shape (few numbered statements at level 1 and each lower level having more numbered statements than the level above it). Documents that have an hour-glass shape (many numbered statements at high levels, few at mid levels and many at lower levels) are usually those that contain a large amount of introductory and administrative information. Diamond shaped documents (a pyramid followed by decreasing statement counts at levels below the pyramid) indicate that subjects introduced at the higher levels are probably addressed at different levels of detail.

SPECIFICATION DEPTH is a count of the number of imperatives at each level of the document. These numbers also include the count of lower level list items that are introduced at a higher level by an imperative that is followed by a CONTINUANCE. This structure has the same implications as the numbering structure. However, it is significant because it reflects the structure of the requirements as opposed to that of the document. Differences between the shape of the numbering and specification structure are an indication of the amount and location of background and/or introductory information that is included in the document.

The NUMBERING and SPECIFICATION STRUCTURAL counts for Sample.txt are are provided by the following tables:

NUMBERING STRUCTURE		SPECIFICATION STRUCTURE	
DEPTH	OCCURRENCE	DEPTH	OCCURRENCE
1	0	1	0
2	12	2	9
3	0	3	0
4	0	4	0
5	0	5	0
6	0	6	0
7	0	7	0
8	0	8	0
9	0	9	0
TOTAL	12	TOTAL	9

ARM REPORT for file Sample.txt on 03-03-05, at 14:11, Page-8

TEXT STRINGS are the number of individual lines of text read by the ARM program from the source file.

UNIQUE SUBJECTS is the count of unique combinations and permutations of words immediately preceding imperatives in the source file. This count is an indication of the scope of subjects addressed by the specification.

Total text strings: 12 Unique Subjects: 7

The ratio of the total for SPECIFICATION STRUCTURE to total lines of text is an indication of how concise the document is in specifying requirements.

Total for SPECIFICATION STRUCTURE: 9
Total text strings: 12

RATIO = 0.75

The ratio of unique subjects to the total for SPECIFICATION STRUCTURE is also an indicator of the specifications' detail.

```
    Unique Subjects:  7
    Total for SPECIFICATION STRUCTURE:  9

    RATIO =   0.7777778

    SOFTWARE
     ASSURANCE
      TECHNOLOGY
       CENTER

                  AUTOMATED REQUIREMENT MEASUREMENT (ARM)

                         DETAIL IMPERATIVE REPORT

                     FOR SPECIFICATIONS CONTAINED IN

                            THE FILE Sample.txt
ARM IMPERATIVES REPORT FOR FILE Sample.txt      03-03-2005     PAGE  2

shall # 1: In Line No. 3, ParNo. 3.1, @ Depth  2
3.1 CPU card's non-volatile RAM SHALL have a 10 year shelf life when stored at room
tempature.

shall # 2: In Line No. 7, ParNo. 7.1, @ Depth  2
7.1 CPU card's Base-R interface SHALL be an RJ-45 connector on the back panel.

shall # 3: In Line No. 8, ParNo. 8.1, @ Depth  2
8.1 CPU Base-R connector SHALL use industry standard pinout.

shall # 4: In Line No. 10, ParNo. 10.1, @ Depth  2
10.1 CPR card's R-chip interface pinout SHALL be TBD.

shall # 5: In Line No. 12, ParNo. 12.1, @ Depth  2
12.1 SPU card SHALL support switch over to a back-up in a normal amount of time.

will # 1: In Line No. 1, ParNo. 1.1, @ Depth  2
1.1 CPU card WILL provide a serial debug interface

will # 2: In Line No. 5, ParNo. 5.1, @ Depth  2
5.1 CPU card WILL provide 16 Mbytes of DRAM (dynamic RAM)

will # 3: In Line No. 6, ParNo. 6.1, @ Depth  2
6.1 CPU card WILL provide 12 Mbytes of program FLASH

will # 4: In Line No. 11, ParNo. 11.1, @ Depth  2
11.1 APU WILL provide 128 Kbytes of surge in a socket.

                  AUTOMATED REQUIREMENT MEASUREMENT (ARM)

                        DETAIL WEAK PHRASE REPORT

                     FOR SPECIFICATIONS CONTAINED IN

                            THE FILE Sample.txt


ARM WEAK PHRASE REPORT FOR FILE Sample.txt      03-03-2005     PAGE  2

normal # 1: In Line No. 12, ParNo. 12.1, @ Depth  2
12.1 SPU card shall support switch over to a back-up in a NORMAL amount of time.
                  AUTOMATED REQUIREMENT MEASUREMENT (ARM)

                         DETAIL INCOMPLETE REPORT

                     FOR SPECIFICATIONS CONTAINED IN

                            THE FILE Sample.txt

ARM INCOMPLETE REPORT FOR FILE Sample.txt         03-03-2005      PAGE  2

TBD # 1: In Line No. 4, ParNo. 4.1, @ Depth  2
```

```
4.1 CPU card's non-volatile RAM can provide TBD days of data storage.

TBD # 2: In Line No. 10, ParNo. 10.1, @ Depth  2
10.1 CPR card's R-chip interface pinout shall be TBD.

TBS # 1: In Line No. 2, ParNo. 2.1, @ Depth  2
2.1 TBS
                    AUTOMATED REQUIREMENT MEASUREMENT (ARM)

                              DETAIL OPTION REPORT

                       FOR SPECIFICATIONS CONTAINED IN

                              THE FILE Sample.txt

ARM OPTIONS REPORT FOR FILE Sample.txt         03-03-2005      PAGE  2

can # 1: In Line No. 4, ParNo. 4.1, @ Depth  2
4.1 CPU card's non-volatile RAM CAN provide TBD days of data storage.

may # 1: In Line No. 9, ParNo. 9.1, @ Depth  2
9.1 CPU RS-485 interface MAY be a 9 pin female d-sub connector located on the front
panel.
```

References

1. [Davis93] A. M. Davis, "Software Requirements: Objects, Functions and States", Prentice-Hall Inc., 1993.

2. [Oulu99] Ilkka Tervonen, "Quality-Driven Assessment: A Pre-Review method for Object-Oriented Software Development", University of OULU, Department of Information Processing Science, Research Papers, Series A19, 1994.

3. [Bray99] M. Bray, M. Ross, and G Staples, "Software Quality Management Fourth, Improving Quality", Mechanical Engineering Publication Ltd., 1996.

4. [EEE84] Institute for Electrical and Electronics Engineers, "IEEE Guide to Software requirements Specifications", Standard 830-1984, New York: IEEE Computer Society Press, 1984.

5. [IEEE93] A. M. Davis and Team, Institute for Electrical and Electronics Engineers, "Identifying and Measuring Quality in A Software Requirements Specification", 1993.

6. [CAR90] Caruso, J. Private Communication, Fairfax, Virginia, fall 1990.

7. [Fenton97] N. E. Fenton and S. L. Pfleeger, "Software metrics A Rigorous & Practical Approach", PWS Publications, 1997.

8. [Fenton91] N. E. Fenton, "Software Metrics A Rigorous Approach", Chapman & Hall, London, 1991

9. [Basili88] V.R. Basili, H. D. Rombach., "The TAME Project: Towards Improvement-Oriented Software Environments", In IEEE Transactions in Software Engineering 14(6) November 1988.

10. [Goodman93] Paul Goodman, "Practical Implementation of Software Metrics", McGraw Hill, London, 1993.

11. [Grady92] Robert B. Grady, "Practical Software Metrics for project Management and Process Improvement", Prentice-Hall, Englewood Cliffs, 1992.

12. [Everald88] Software Metrics SEI Curriculum Module SEI-CM-12-1.1, Carnegie Mellon University Software Institute, Dec 1988.

13. [IEEE96] M. Chistensen, N. Grumman, C. Chang, “Blueprint for the ideal Requirements Engineer”, University of Illinois, March 1996.

14. [Capers94] Caper Jones, “Software Metrics”, Software Productivity Research Inc, September 1994.

15. [IEEE98] D. M. Berry, B. Lawrence, “Requirements Engineering”, IEEE Software, March 1998.

16. [Costello-Liu- 95] R. J. Costello, D. B. Liu, “Metric for Requirements Engineering”, Elsevies Science Inc, 1995.

17. [IESE03] M. M. Mora, C. Denger, “Requirements Metrics”, IESE-Report No. 096.03/ Version 1.0, October 1, 2003.

18. [SATC97] W. M. Wilson, L. H. Rosenberg and L. E. Hyatt, “Automated Quality Analysis of Natural Language Requirement Specifications”, ISCE, May 1997.

19. [REDTP] Ganska, Ralph, John, Rubinstein, Jack, Van Buren and Jim, “Requirement Engineering and Design Technology Report”, Software Technology Support Center, Hill Air Fource Base.

20. [IEEE93] IEEE, “Recommended Practice for Software Requirements Specifications”, IEEE Computer Society, December 2, 1993.

21. [Wiley02] G. Kontonya and I. Sommerville, “Requirements Engineering Process and Techniques”, Wiley Publications, Reprinted February and October 2002.

22. [NCOSE95] R. Stevens and J. Martin, “What is Requirements Management?” NCOSE Requirements Management working group, 1995.

23. [Loconsole03] A. Loconsole and J. Borstler, “Theoretical Validation and Case Study Of Requirements Management Measures”, 2003.

24. [Omni99] A. M. Davis, Omni-Vista, D. A. Leffingwell, “Making Requirements Management Work for you”, Rational Software Inc., April 1999.

25. [Linda95] Dr. L. H. Rosenberg and L. E. Hyatt, “Developing a Successful Metrics Program”, presented at the International Conference on Software Engineering, Wednesday April 24, 1995.

27. [Nguyen 02] L. Nguyen and P. A. Swatman, “Managing the requirements engineering process”, April 2002.

28. [Fine-Grain] Fine-Grain Process Modeling for Collaborative Work Support: Experience with CPCE by Jacques Lonchamp, Bruno Denis. http://www.loria.fr/~jloncham/jds.pdf, Last access Feb 2005.

29. [GDSM96] R. E. Park, W. B. Goethert, W. A. Florac, "Goal-Driven Software Measurement, A Guidebook", Aug 1996. Handbook CMU/SET-96-HB-002.

30. [Linda-Lawrence97] Dr. L. H. Rosenberg and L. E. Hyatt, "Developing a Successful Metrics Program", presented it at International Conference on Software Engineering, San Fransisco, CA, November 1997.

31. [Wiegers99] K. E. Wiegers, "Writing Quality Requirements", published in Software Development Magazine, 1999.

31. [Paul] Dr. Paul Dorsey, Top 10 reasons why systems projects fail.

32. [SPC] Software Productivity Center Inc. to achieve business goals through effective software development practice. http://www.spc.ca/resources/metrics/intro.htm Last access Feb 2005.

33. [Nick99] M Nick, K D Althoff, C Tautz, "Facilitating the Practical Evaluation Organizational Memories Using the Goal-Question-Metric Technique", Fraunhofer Institute for Experimental Software Engineering Sauerwiesen 6, D-67661 Kaiserslautern, Germany, 1999 http://www.iese.fhg.de/pdf_files/althoff_pub/kaw99-crc.pdf, Last access Feb 2005.

34. [GQM98] A. Fuggetta and Team, "Applying GQM in an Industrial Software Factory", ACM Transactions on Software Engineering and Methodology, Vol 7. No. 4, October 1998.