Proceedings of the 5th National Conference; INDIACom-2011
Computing For Nation Development, March 10 – 11, 2011
Bharati Vidyapeeth's Institute of Computer Applications and Management, New Delhi

Metrics for Requirements Engineering and Automated Requirements Tools

Mohammad Ubaidullah Bokhari[1] and **Shams Tabrez Siddiqui**[2]
[1, 2]Department of Computer Science, AMU, Aligarh
[1]mubokhari@gmail.com and [2]ksamtab@rediffmail.com

ABSTRACT
Software requirements are the foundations from which quality is measured. Measurement enables to improve the software process; assist in planning, tracking and controlling the software project and assess the quality of the software thus produced. Quality issues such as accuracy, security and performance are often crucial to the success of a software system. Quality should be maintained from starting phase of software development. Requirements management, play an important role in maintaining quality of software. A project can deliver the right solution on time and within budget with proper requirements management. Software quality can be maintained by checking quality attributes in requirements document. Requirements metrics such as volatility, traceability, size and completeness are used to measure requirements engineering phase of software development lifecycle. Manual measurement is expensive, time consuming and prone to error therefore automated tools should be used. Automated requirements tools are helpful in measuring requirements metrics. The aim of this paper is to study, analyze requirements metrics and automated requirements tools, which will help in choosing right metrics to measure software development based on the evaluation of Automated Requirements Tools

KEYWORDS
Software requirements, Software quality, Software development, Requirements management, Quality attributes, Requirements metrics, Automated Requirements Tools

1. INTRODUCTION

In the last few decades, software projects have encountered major difficulties. Most software engineering projects tend to be late and over budget. Several of the causes of these failures are related to requirements engineering issues such as requirements creep, poorly documented requirements, requirements that were impossible to satisfy, and requirements that failed to meet the needs of the user. Good requirements management practices help improve customer satisfaction, lower the system development costs and increase the chance of having successful project. Requirements metrics when incorporated in requirements management, it assists in analyzing the quality of requirements and identifying the reasons for software re-engineering/failure. Requirements metrics define the output measures of the software process.

Requirements engineering helps software engineers to better understand the problem they will work to solve [5]. Requirement engineering begins with inception-a task that defines the scope and nature of the problem to be solved. Its moves onward to elicitation- a task that helps the customer to define what is required, and then elaboration- where basic requirement are defined and modified. As the customer defines the problem, negotiation occurs- what are the priorities, what is essential, when it is required? Finally the problem is specified in some manner and then reviewed or validated to ensure that a stakeholder understanding of the problem and the customer's understanding of the problem coincides.

Collecting requirements is not an easy task. Requirements engineering has critical problems which can be due to lack of stakeholder's involvement in the requirements process. Lack of requirements management skills also leads to bad requirements engineering. Because of the natural human tending to hide bad news, the real statistic may be even higher [17]. Unclear responsibilities and communication among stake holders can also lead to bad requirements engineering.

Functional requirements or behavioral requirements define functions of the product. Functional requirements include input that the software gets and output it generates. Non-Functional requirements or non-behavioral requirements are the properties of software such as portability, reliability, reusability, testability, efficiency and modifiability [6].

Requirements management is the process of eliciting, documenting and communicating requirements. Requirements management play important role in success of software. It manages changes to requirements and maintains traceability in requirements documents. Requirements can be written using quality attributes known as software requirements specification. Success rate of product depends on process used by organization. Every company needs to assess their present approach in order to remain competent in dynamic market. Capability Maturity Model, Goal Question Metrics, BOOTSTRAP, The ISO 9000 process improvement models are used to assess process and suggest methods to improve them. Once new process are adopted their performance should be checked, therefore measurement of software is necessary. Software can be measured using process, product, resources and requirements metrics.

Requirements metrics are useful in identifying risks of a project by identifying errors in requirements document. There are many metrics used for measuring the requirements. Volatility metrics checks the changes of the requirements, traceability evaluates links between requirements to requirements within a document and requirements completeness metrics checks whether the requirements specified and complete or not. These metrics validate the written requirements against actual requirements. Single metric cannot ensure overall quality therefore multiple metrics should be used for measurement. Measuring the metrics manually is a tedious and time consuming task. Moreover it is not free from errors. Therefore automated requirements measurement tools should be used. With the use of automated requirements tools collection of metrics is faster and more reliable.

Requirements can be managed in a better way through automated tools and consequently better requirements management leads to better product. Automated Requirements Tool, Dynamic Object Oriented Requirements Systems, Requirements Use Case Tool and IBM Rational Rose are some prominent automated requirements tools used in measuring software, collecting, viewing and changing requirements.

2. REQUIREMENTS ENGINEERING

As one of the processes in software engineering, requirements engineering (RE) plays a vital role in ensuring the overall success of the software engineering process. Requirement engineering is the branch of software engineering concerned with the real-world goals for, functions of and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software engineering behavior and their evolution overtime across software families [16].

Requirements engineering is a process of discovering the needs of stake holders and documenting them for analysis, communication and implementation [2]. Many errors can be detected in the requirements phase. Davis [1] claims that fixing of errors detected in later stages of software development is more expensive than the initial stages. If errors are not detected in the requirements phase it leads to wrong product development. Wrong requirements can also lead to wastage of valuable resources.

Requirements are developed through requirements engineering. Requirements engineering is a process which include a set of activities such as:

- Problem synthesis [9]
- Requirements Elicitation
- Requirements Analysis and Negotiation
- Requirements Specification
- System modeling [33]
- Requirements Verification and Validation
- Requirements Documentation [23]
- Requirements Management

2.1 PROBLEM SYNTHESIS

Synthesis involves the ability of putting together the parts you analyzed with other information to create something original. You reach out for data or ideas derived from a variety of sources [10].

Requirement Synthesis is a tool for the automatic synthesis of the requirement automata for safety properties. Requirement automata make assumptions for the behavior of a component. Program documentation is a typical process [11], where as synthesized requirements help the user to understand the interaction of the program components; program certification, re- verification of the system(possibly by a different user and different tool); and compositional model checking run, in order to avoid the construction of the full product state space.

Requirement synthesis supports three different operation modes: assumption generation, compositional model checking, and front-end to the model checker SPIN. In assumption generation mode, Requirement Synthesis minimizes the sizes of the assumption; small assumptions are useful for program documentation and as certificates for re-verification. In compositional model checking mode, Requirement Synthesis terminates as soon as the property is proven or disproven, independently of the size of the assumption. In front-end mode, Requirement Synthesis terminates when the size of the assumption falls below a specified threshold and calls SPIN with the simplified verification problem.

2.2 REQUIREMENTS ELICITATION

Requirements elicitation is a process of identifying needs and bridging the disparities among the involved communities for the purpose of defining and distilling requirements to meet the constraints of these communities [29]. Other often used names for requirements elicitation are requirements acquisition, requirements capture, requirements discovery, requirements gathering, problem analysis, understanding etc. This process requires application domain knowledge, organizational knowledge and technical knowledge, as well as specific problem knowledge. Requirements elicitations are non-trivial because you cannot get all requirements from user and customer by just asking what a system should do.

A lot of techniques can be used in the requirements elicitation process [19].Some examples are:

- Interviews. This is the most commonly used technique.
- Ethnography. This technique is adopted from sociology and includes the observation of users.
- Questionnaire. Question is prepared with limited choice of options.
- Contextual query. This is a knowledge acquisition approach.
- Scenarios, use cases and goal-based techniques. These techniques can only be used in the later stages of requirements elicitation.

2.3 REQUIREMENTS ANALYSIS AND NEGOTIATION

Requirements Analysis and Negotiation is a process during which requirements are analyzed and modeled. Possible conflicts are resolved by negotiation between stakeholders. The elicitation process provides the input to this process. The output of the process is a consistent and complete set of requirements.

Some typical techniques that can be used during this phase are:

- Unified Modeling Language (UML). UML is a collection of techniques for modeling software or systems.

- Specification and Description Language (SDL). SDL is a relatively mature requirement description and definition language [31].
- Structured Analysis Structured Design (SASD). SASD uses a set of different techniques such as functional decomposition technique, data-flow diagram and data dictionary [31].
- Goal-based techniques. These techniques can be used in the later stage of requirements elicitation as well as very effective requirements analysis techniques [27, 20, 26].
- Petri Nets. Petri Nets are good at modeling state transitions and control flow in an asynchronous system where there is a lot of parallel and asynchronous events [28].

2.4 REQUIREMENTS VERIFICATION AND VALIDATION

Requirements verification and validation (V&V) is the process to ensure that the requirements document is unambiguous, consistent and complete, that the stakeholders are satisfied with the final requirements specification. SRS can be used as guidance for requirements verification and validation.

The output of the requirements documentation process is the input of the verification and validation process. The output of the V&V process is the finalized requirements specification document agreed and authorized by all stakeholders.

The techniques used most often for this process are:

- Formal requirements inspection. Inspection is used for identification of incompleteness, inconsistency, ambiguity and conflict in the requirements specification [24].
- Requirements testing. This technique is used to examine the implementability and understandability of the requirements through writing test cases for requirements [23].
- Requirements checklist. This technique is used to examine a list of common concerns in requirements specifications [23].

2.5 SYSTEM MODELING

System modeling is a technique to express, visualize, analyze and transform the architecture of a system. Here, a system may consist of software components, hardware components or both and the connections between these components. A system model is a skeletal model of the system.

System modeling is intended to assist in developing and maintaining large systems with emphasis on the construction phase. System modeling can increase reliability and reduce development cost by making it easier to build systems, to reuse previous built components within new systems, to change systems to suit changing requirements such as functional enhancement and platform changes to understand systems [13].

For system modeling, we need a conceptual framework and a system modeling language, textual and/or diagrammatic. Besides textual notations like tables or prose, diagrammatic notations like graphs are common today. Within these diagrams, there are symbols representing the parts of the system e.g. objects and groups of objects, and other symbols visualizing the connections between these parts. The number of symbols for each purpose differs noticeably between notations.

During the past decades four main conceptual frameworks have evolved:

- Design methods. Design methods consist of a concept, a language and a design process.
- Module Interconnection Languages. Module interconnection languages (MILs) are means to support the connection effort of large systems.
- Software Architectures. The system designer use Design methods and Module interconnection languages for low level of abstraction. Software architectures manages increasing sizes, complexity, specification and design of the overall system structure.
- Design Patterns. Design patterns are the most theoretical approach to model a system and its behavior. There exists no formal definition of the term ‘design pattern’, but the smallest common denominator seems to be ‘a reusable solution to a problem in a particular context’.

2.6 SOFTWARE REQUIREMENTS SPECIFICATION

The Requirements document or Software Requirements Specification describes external behavior of software system. This document can be written by user/customer or developer. A well written requirements document helps in meeting the desired goals of successful software within time limit. Errors that are found in the requirements document are easy to fix, cost less and consume less time. If these documents are handled properly errors can be reduced. There are two types of errors that can be found in requirements document. Knowledge errors, which are caused due to not knowing what the requirements are and Specification errors, caused due to lack of knowledge or experience of specifying requirements.

INTERNAL attributes of requirements documents describe how requirements should be specified. What they should include and how they effect others attributes.

Unambiguous - A requirement document should be unambiguous but natural language such as English has many inherent ambiguous words. Deterministic Finite State Machine (FM), Pertinens, and Decisional Tree are used for formal specification because they have less ambiguity inherent in them [8].

Correct - A requirements document is correct if and only if every sentence or requirement mentioned in the document is build by the system.

Complete - A requirement document is said to be complete if it satisfies 4 conditions.

1. The document should include each and everything that the software supposed to do [1].
2. All pages, tables and figures are numbered. Names and references should be present in referenced material.
3. In the document no ‘TB’ To Be Determined.
4. There should be some output for every input of the system.

Understandable - A requirement document should be understandable by all readers which include customers, users, project managers etc.

Verifiable - A requirements document is verifiable if there exist finite cost effective process with which a person or machine can check that the built software product meets the requirements.

Internal Consistent - A requirements document is said to be internally consistent if no subset of requirements conflict such as focus, drag and press with other requirements of a set.

Modifiable - A requirements document is said to be modifiable if changes can be made consistently. Improvements and removing errors are also important for modification of the documents.

Annotated by Relative Importance - A requirements documents is said to be annotated by relative importance if its features are in ascending order according to there importance to the customers. They can be marked as compulsory, recommended and optional [8].

Annotated by Relative Stability- A requirements documents can be annotated by relative stability depending on the help by designers. Changes of requirements is the cause of the stability of the features like which can be prone to early changes or which features can stay for longer period.

Annotated by Version - A requirements documents is said to be annotated by version if the reader can determine which features can be implemented in which version of the software. This attribute can be application dependent.

Precise - A requirements document is precise if it should not contain vague details. For example the system should not keep the user waiting for acknowledgment but it should have certain time like 2 minutes.

Traced - A requirements document is traced if the origin of each of its requirements have references to earlier versions of the supportive documents see figure 2.6.

Traceable - A requirements document is said to be traceable if it has reference of every requirement stated [1]. This can be achieved by numbering every paragraphs and requirements uniquely.

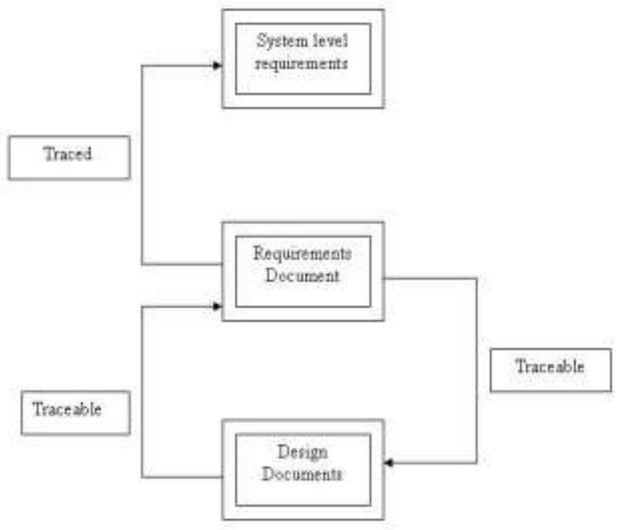


Figure 2.6 : Traced Vs Traceability (according to [34])

Not Redundant - If the requirements document has same information stored in more than one place it is called redundant. Redundancy is helpful for readability but it can also create problems. Change at one place not at the other places leads to inconsistencies.

At right level of detail - A requirements document provides different levels of details. It can be from very general to very specific. It should cover the minimum functions specified by the customer. For example, System should accept payment, cash payment and previous bills with credit cards.

Organized - A requirements document is organized if its contents are made for easy navigation of information to users. Grouping the functional requirements by user class, object and features are some methods that make the document organized [8].

EXTERNAL attributes of requirements documents describe overall or outer appearance of SRS.

Achievable - A requirements document is achievable if there exists at least one system design and implementation that correctly implements all the requirements stated in the document. This can be done by building prototypes of the system [8].

Electronically Stored - If a word processor is used for storing the requirements documents then it is electronically stored [8]. It can be generated from requirements database.

Concise- A requirements document is said to be concise if the length of the document is made short without effecting other qualities.

Design Independent - Requirements document should be design independent. No single solution should be made compulsory if there are many ways to solve a problem. Only required features should be mentioned instead of imposing a design.

Reusable - A requirements document is said to be reusable when it is used for the future requirements documents which is helpful in reducing time. Reusability could be of general terms in non similar projects and specialized for similar projects.

2.7 REQUIREMENTS DOCUMENTATION

Requirements documentation is the process to select proper notations to document requirements at the appropriate level of detail. The document structure and the labeling hierarchy of requirements are very closely related to requirements management.

The documentation process receives its input from the analysis and negotiation process. The output of the process is a well-structured and defined specification, which, however, still needs to be verified and validated.

Requirements can be modeled in informal (natural language), semi-formal (diagrams, graph) and formal languages (Mathematical notations), therefore, the requirements documentation might contain models in any of the three different notations or combinations of them. Even if the system is documented in a formal notation, an informal document is usually also required to improve understandability of the SRS.

2.8 REQUIREMENTS MANAGEMENT

Requirements Management is a process of managing changes to requirements. Changes are inevitable because of system errors and better understanding development of customers' real needs. Requirements Management is carried out in parallel with other requirements activities in the software development. Activities of requirements management includes keeping project plan undated with requirements, controlling requirements versions, tracking status of requirements and tracing the requirements as depicted in figure 2.8.

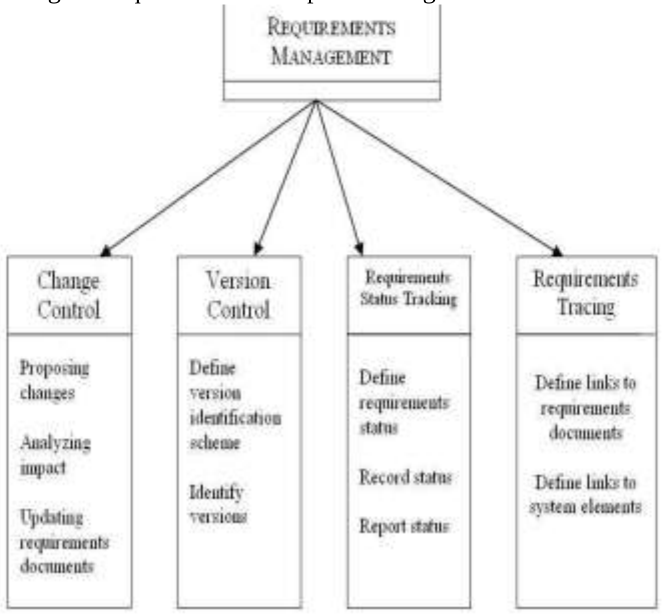

Requirements Management
Change Control
Proposing changes
Analyzing impact
Updating requirements documents
Version Control
Define version identification scheme
Identify versions
Requirements Status Tracking
Define requirements status
Record status
Report status
Requirements Tracing
Define links to requirements documents
Define links to system elements

Figure 2.8: Requirements Management Activities (according to [35])

Every project is prone to changes during its development. Changes to requirements cannot be totally avoided but they can be managed. Changes can be due to addition of requirements, fixing of errors, customers new needs etc. There could be environmental changes for example platform changes from Windows to Linux. In order to maintain changes large database should be maintained and tracked on requirements document.

If database about changes to requirements is not maintained than developers might work on the changed requirements and resources can be wasted. Links should be maintained to changed requirements. Database record identification and symbolic identification can also be used for numbering the requirements. Requirements must be traceable and traced to customers' needs.

3. SOFTWARE PROCESS ASSESSMENT

Once a software process model is developed, the criteria for judging the quality of the model have to be established. To this end, the Capability Maturity Model (CMM) was developed by the Software Engineering Institute (SEI) in the mid-1980s. It was initially developed as a software process assessment model but can also serve as a tool to improve the software process within an organization. CMM defines five levels of maturity of organizations:

(1) Level 1-Initial: At this maturity level, processes are ad hoc and chaotic. Success in organizations at Level 1 depends on the competence and heroism of the people in the organization and not on the use of proven processes. Such organizations are characterized by a tendency to over commit, a tendency to abandon processes in times of crisis, and eventually failure of projects.

(2) Level 2- Repeatable: Rudimentary software development processes are repeatedly used. Organizations at this level use basic project management to track cost and schedule. When these practices are in place, projects are performed and managed according to their documented plans.

(3) Level 3- Defined: At maturity level 3, software processes are well characterized, understood, and described.

(4) Level 4- Managed: Organizations at this level can use precise measurements to effectively control the software development effort. In particular, management can identify ways to adjust and adapt the process to particular projects in order to improve software quality.

(5) Level 5- Optimizing: This level focuses on continually improving process performance through both incremental and innovative technological improvements. Quantitative process improvement objectives for the organization are established, continually revised to reflect changing business objectives and used as criteria in managing process improvement.

CMM has become a widely accepted international standard for software process assessment and improvement. However, several other software process assessment models have also been developed. Some of them are listed below together with key references:

- ISO 9001- A software process assessment model from management science. It is part of the overall suite ISO 9000 [32].
- BOOTSTRAP – A European version of CMM. Several new rating methods for software practices are used in the model [21].
- ISO/IEC 15504 Model – A model with more advanced ratings of and calculations for software practices. It is also called the Spice Model [30].
- CMMI –CMMI stands for Capability Maturity Model Integration (CMMI). It combines System Engineering and Software Engineering models into one integrated model which can be used for multi-purpose assessment.

It is designed to be consistent with ISO/IEC 15504 with some modified assessment models. It includes a staged representation model and continuous representation model. It can also be used to support software process improvement [25]. A detailed discussion about these models can be found in [22].

Software process assessment is important because it is a means for organizations to identify strengths and weaknesses in their existing practices, and key areas for improvement [22].

4. SOFTWARE METRICS

Software metric is a measurement derived from a software product, process, or resource. Its purpose is to provide a quantitative assessment of the extent to which the product, process, or resource possesses certain attributes. 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 [7] simple, objective, easily obtainable, valid and robust.

The goals of software metrics are to identify and measure essential factors which effects software development [3]. Quality attributes discussed in SRS have formula to measure them Table 4.1 shows quality attributes and metrics to measure them.

QUALITY ATTRIBUTES	**METRIC**	**PURPOSE AND REFERENCE VALUE**
Unambiguous	$Q_1 = \frac{n_{ui}}{n_r}$ where, n_{ui} is the number of requirements for which all reviewers presented identical interpretations. n_r is the total number of requirements	To obtain the percentage of requirements that has been interpreted in a unique manner by all its reviewers. Reference value: Close to 0, Ambiguous requirements. Close to 1 , Unambiguous requirements.
Correct	$Q_2 = \frac{n_c}{n_c * n_{NV}} = \frac{n_c}{n_r}$ where, n_c is the number of correct requirements n_{NV} still not valid requirements n_r is the total number of requirements	To measure percentage of requirements in the SRS that has been validated. Reference value: 0= incorrect. 1= correct.
Complete	$Q_3 = \frac{n_u}{n_i * n_s}$ where, n_u is the unique function n_i is the stimulus input of the function n_s is the state input of the function	To measure the total number of functions currently specified. Reference value: Grows closer to 1, the more complete it becomes .
Understandable	$Q_4 = \frac{n_{ur}}{n_r}$ where, n_{ur} is the number of requirements understood by all reviewers. n_r is the total number of	To measure the number of requirements those are understood by all users (reviewers). Reference value: 0= No requirement understood. 1= All requirements understood.

	requirements	
Verifiable	$Q_5 = \frac{n_r}{n_r + \sum_i c(r_i) + \sum_i t(r_i)}$ where, n_r is the total number of requirements c is the cost necessary to verify presence of requirements t is the time necessary to verify presence of requirements	To measure the verifiability of a requirement. Reference value: 0= Very poor. 1= Very good.
Internal Consistent	$Q_6 = \frac{n_u - n_n}{n_u}$ where, n_u is the number of unique functions specified n_n is the number of unique functions that are non deterministic	To measure the percentage of unique functions that are deterministic assuming that SRS can be defined as a function that maps inputs and states into outputs.
Modifiable	Not defined.	To measure if- it contains a table of context and index. Reference value: 1= If the table of context and index. 0= Otherwise.
Annotated by Relative Importance	Not defined.	To calculate the percentage of requirements that are annotated by relative importance.
Annotated by Relative Stability	Not defined.	To calculate the percentage of requirements that are annotated by relative stability.
Annotated by Version	Not defined.	To calculate the percentage of requirements annotated by version.
Precise	$Q_7 = \frac{n_p}{n_p + n_f}$ where, n_p is the number of true positives n_f is the number of false positives	provide minimum time for acknowledgment
Traced	Not defined.	Measuring the level of trace-ness is impossible, therefore none metric is presented. Reference value: Not defined.
Traceable	Not defined.	To measure which requirements are being supported by the components or verified by testing. If a single requirement is not traceable, then the whole SRS is untraceable. Reference value: 1= Traced attributes. 0= Otherwise.
Not Redundant	$Q_8 = \frac{n_f}{n_u}$ where,	To measure the percentage of unique functions that are not repeated. Reference value:

	n_f is the total functions currently specified. n_u is the current unique functions specified	1= No redundancy. 0= Complete redundant.
At right level of detail	Not defined.	Is not possible to measure. Reference value: Not defined.
Organized	Not defined.	Not defined.
Achievable	Not defined.	The measure of the existence of a single system. Reference value: 1= A set requirement are either achievable. 0= Given acceptable development resources the SRS is achievable.
Electronically Stored	Not defined.	To measure the percentage of the volume of the SRS that has been electronically stored. Reference value: 1= Electronically stored. 0= Otherwise.
Concise	$Q_9 = \frac{1}{size+1}$ where, size is the number of pages.	If two SRS documents are equal , we can choose best one using this formula. Reference value: 1= Best SRS 0= Worst SRS.
Design Independent	$Q_{10} = \frac{D(R_E \cup R_I)}{D(R_E)}$ where, R_E is the total requirements that describe pure external behavior. R_I is the total requirements that directly address architecture or algorithms of the solution.	To meet the percentage of possible solution systems that are eliminated by adding the overly constraining requirements. Reference value: 1= Design Independent 0= Highly Design Independent
Reusable	**Not defined.**	To measure if SRS has been reused. Reference value: 1= SRS is reusable. 0= Not reusable.

Table 4.1 SRS with metrics count

5. REQUIREMENTS METRICS

Requirement engineering deals with elicitation, analysis, communication and validation of the requirements. As early as errors are identified it is easy to fix them compared to later identification of errors. It is obvious that the cost of fixing these errors in initial stages is lower than fixing them in later stages of software development. Metrics that can be gathered from requirements are the size of the requirements, requirements traceability, requirements completeness and requirements volatility.

5.1SIZE METRICS

Size is an important and general metrics used to measure requirements. LOS is common metrics used to measure size of software. Comment lines and blank lines are not considered as code. A Non commented line is considered as a code but it also include executable commands, executable statements and data declarations. Use Cases can also be considered as a size measurement when they are used to describe requirements. For example number of use cases, number of functions covered etc. Use case metrics are categorized into 3 categories [15].

- Size metrics. Size metrics includes number of atomic actions in main flow. Size of use case can be determined by counting number of actors weighted by their complexity.

- Environment metrics. Environment metrics are the factors which has influence on complexity level of the use cases. These are independent of the size of the use cases.
- Composite metrics. Composite metrics are derived from size metrics and environment metrics.

5.2 REQUIREMENTS TRACEABILITY METRICS

Traceability is the ability to trace requirements in a specification to their origin from higher level to lower level requirements in a set of documented links. Traceability provides information which helps in determining whether all relationship and dependencies are addressed. This also makes requirements leakage - existence of lower level requirements with no valid origin from higher level visible. It helps in preventing wrong interpretations of other metrics. There exist 5 types of traceability metrics [3].

- Next level coverage metrics - COB. This metric traces number of requirements to next level up and next level down. This metrics also trace number of requirements in both directions.
- Full depth and height coverage - DECO. This is similar to COB and best suited for 3 or less level specifications. It traces requirements to highest and lowest level of specifications instead of only immediate next level in both directions.
- Inconsistent traceability metrics - This include the numbers of requirements in a specification that have ablest one inconsistent link in both upward and downward directions.
- Undefined traceability metrics - It include the numbers of requirements in a specification that have no traceability links upward and download. However, the highest link will not have any upward link as it is derived from design document and the lowest level link has no down link.
- Linkage statistic metric- It measure the complexity level of traceability data by counting the number of higher or lower level requirements to which each requirements specification is traced.

5.3 REQUIREMENTS COMPLETENESS METRICS

Requirements Completeness Metrics are used to assess whether a requirement is at wrong level of hierarchy or too complex. Requirements are considered as complete if all pages are numbered, all figures and tables have captions and numbers, all references are present, all sections and subsections are numbered and no 'TB' to be determine should be present see SRS for details. Requirements Completeness Metrics are of 3 types [3].

- Requirements Decomposition Metrics - This metric is used to know whether the requirements specified have been decomposed sufficiently to be used in next phase. A complex function will have many levels of specification and a simple function will have few levels.
- Requirements Specification Development - This metric is used to provide information about the work completed and the work remaining for a given specification.
- Requirements Specification Metrics - This metric provide information about the quantity of items for which issue resolution is needed. This metric used 'to be determined', 'to be supplied' tags used for completing the requirements document.

5.4 REQUIREMENTS VOLATILITY METRICS

The degree to which requirement changes over a time period is called volatility of requirements. This metric is used to assess reasons for change of requirements over a period of time. Requirements degree of change is also checked by this metric. Both these factors are checked to know whether the changes are consistent with current development activities. It indicates changes such as addition, deletion and modifications. It helps in tracing future requirements, design and code volatility. Volatility can be high in the initial phase of software development. It should be reduced as the project progress so that further development should not be affected. This metric is a good indication of other metrics for example, if a specific portion of software is tested without knowing the volatility of the requirements than this testing would not be wrathful.

6. REQUIREMENTS TOOLS

Measuring the metrics manually is a tedious and time consuming task. Moreover it is not free from errors. Therefore automated requirements measurement tools should be used. Collection of metrics is faster and more reliable with the use of automated requirements tools. Requirements can be managed in a better way through these automated tools and consequently better requirements management leads to better product. Automated Requirements Tool, Dynamic Object Oriented Requirements Systems, Requirements Use Case Tool and IBM Rational Rose are few automated requirements tools used for collecting, viewing, and changing requirements are discussed below.

6.1 AUTOMATED REQUIREMENTS MEASUREMENT TOOL – ARM

The Automated Requirement Measurement (ARM) Tool was developed by the Software Assurance Technology Center (SATC) at the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center. It is an early life cycle tool for assessing requirements that are specified in natural language. The Objective of the ARM tool is to measure the quality or requirements documents/SRS. It helps in "Writing the requirements right", not "Writing the right requirements". ARM tool basically fetch quality indicators from requirements documents. ARM tool creates a file that includes three reports. This file contains a summary report, imperative report and detail weak phrase report.

NASA categorized quality indicators in two types of classes .They are in relation to individual statements and in relation to total requirements documents [4].

Imperatives

Imperatives are the words and phrases that commands that something must be provided such as shall, must or must not, is required to, are applicable, responsible for, will and should.

Directives

Directives are the words and phrases that point to illustrative information within the requirements document. The data and information pointed by directives are more understandable and they are figure, table, for example and note.

Options

Options are the words that help in reducing the overall control over final products and allow changes in possible cost and schedule risks. The words are May, Can and Optionally.

Continuances

Continuances are phrases which follow an imperative and introduce the specification of requirements at lower level. Continuances are used for an indication that requirements were organized and structured. Traceability is done using these phrases such as below, as follows, Following, listed, in particular and support.

Weak phrases

The weak phrases such as adequate, easy, as a minimum, be able capability to, capability of, timely, normal provide for effective etc are an indication of the extent that the specification is ambiguous and incomplete.

Text Structure

Text Structure is the number of statement identifiers found at each hierarchical level of the requirements document. The text structure of documents judges to be well organized.

Specification Depth

Differences between the Text Structure counts and the Specification Depth were found to be an indication of the amount and location of text describing the environment that was included in the requirements document.

Size

Size has three indicators such as lines of text, imperatives and subjects of specification statements. The imperatives provide an indication of how concise the document is specifying the requirements.

Readability Statistics

Readability Statistics measures how easily an adult can read and understand the requirements document.

Requirements Use Case Tool (RUT)

The Requirements Use case Tool (RUT) helps managers, customers, and developers in assessing the quality of use cases. The RUT provides a series of metrics useful for calculating information about the relationships among the captured use cases and performs use case evaluation by searching text and identifying risk indicators such as incomplete or weak phrases. For collecting, evaluating, and maintaining software requirements RUT is a valuable resource. RUT combines with the industry-standard tool developer Rational Rose, for developing UML diagrams.

RUT is a web-based, multi-user application that provides the ability to create, view, and modify use cases and related information for a particular project. The tool was developed for using multi-platform, open source technologies (PHP and MySQL).

Furthermore, this development approach provides users the ability to modify the system to suit their individual project needs and will simplify the process of adding additional features or components to the tool in the future. To ensure consistency, RUT provides a standard use case template [12] to be used for all use case entry into the repository.

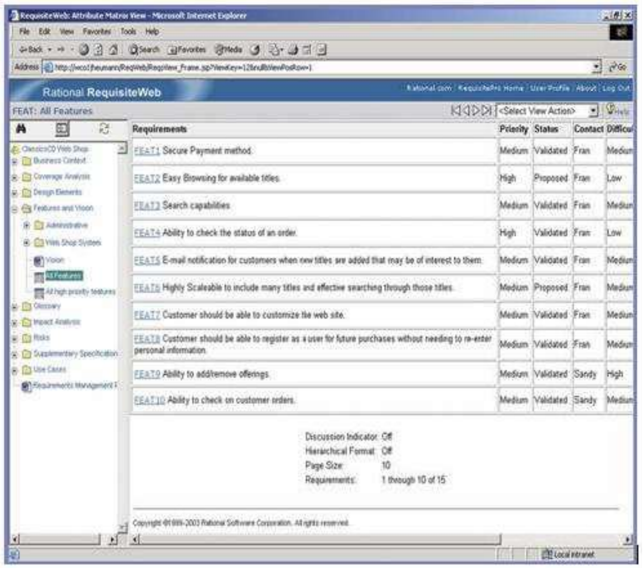
One of the most beneficial features of RUT in terms of quality analysis is its capability to parse use case text for matches against a pre-defined and user-modifiable set of risk indicators. An effort is currently underway to transition this tool into general usage among NASA's software project teams.

6.2 IBM RATIONAL REQUISITEPRO

IBM Rational RequisitePro is a requirements management tool that uses Microsoft word as the primary user interface for managing requirements. It helps in writing good use cases, improves the traceability and manages the requirements. Rational RequisitePro increases the quality, reduces the risks and lead to right solution on time and within budget.

Rational RequisitePro supports databases such as Oracle, Microsoft SQL Server and Microsoft Access. The tool is aimed at novice users who have less experience in requirements management tools. The tool is only suitable for small and medium sized projects. The tool supports change impact analysis, bi-directional traceability and cross-project traceability.

Software development is a team endeavor, so it is important that the team members share understanding, vision, goals, specifications and requirements of the project. Change tracking is a powerful feature of IBM Rational tool. It manages change by linking requirements so that when changes occur to one requirement its effect is seen on other requirements. Each modification is captured, tracked and documented. The IBM Rational Requisitely has a web interface see figure 5.1. It makes requirements accessible from multi platform and remote locations. This tool also makes developers, managers and testers accessible for changes and communication.



RequisiteWeb: Attribute Matrix View - Microsoft Internet Explorer

Rational RequisiteWeb

FEAT: All Features

Requirements	Priority	Status	Contact	Difficu
FEAT1 Secure Payment method.	Medium	Validated	Fran	Medium
FEAT2 Easy Browsing for available titles.	High	Proposed	Fran	Low
FEAT3 Search capabilities.	Medium	Validated	Fran	Medium
FEAT4 Ability to check the status of an order.	High	Validated	Fran	Low
FEAT5 E-mail notification for customers when new titles are added that may be of interest to them.	Medium	Validated	Fran	Medium
FEAT6 Highly Scaleable to include many titles and effective searching through those titles.	Medium	Proposed	Fran	Medium
FEAT7 Customer should be able to customize the web site.	Medium	Validated	Fran	Medium
FEAT8 Customer should be able to register as a user for future purchases without needing to re-enter personal information.	Medium	Validated	Fran	Medium
FEAT9 Ability to add/remove offerings.	Medium	Validated	Sandy	High
FEAT10 Ability to check on customer orders.	Medium	Validated	Sandy	Medium

Discussion Indicator: Off
Hierarchical Format: Off
Page Size: 10
Requirements: 1 through 10 of 15

Copyright ©1999-2003 Rational Software Corporation. All rights reserved.

Figure 5.1: IBM Rational RequisitePro (reprinted courtesy of [37])

6.3 DYNAMIC OBJECT ORIENTED REQUIREMENTS SYSTEM (DOORS)

Teleologic's Dynamic Object Oriented Requirements System - DOORS is a leading market software metrics tool. It supports communication to reduce project failure risk, collaboration to increase productivity and traceability to enable validation. DOORS provide ability to manage changes and exchange requirements data with other requirements management tools. Telelogic has 4 types of tools requirements management.

DOORS help in managing requirements, analyzing requirements, tracing requirements, parsing of requirements and capturing the requirements.

DOORS/Analyst [36] is a requirements management tool which is traceable, easy-to-learn and it also maintains data integrity. UMP-based requirements modeling has a capability for drawing models, pictures, and diagrams inside DOORS see figure 5.2. Features of DOOR Analyst include visual model of use cases capturing, create a common understanding among team members and It provide UMP drawing tool linked with other DOOR requirements tools.

DOORS XT enhances requirements management by better communication for the projects based on distributed multiple sites globally. It has 3-tier architecture which includes DOOR server, Oracle server and web application server. It maintains a central database and provides necessary files to its multiple clients distributed all over the world. Storing database in a central repository reduce installation and maintenance cost. It has the same traceability features as compared to DOOR which helps user to use interface for required information.

DOORSTEP requirements management tool is suitable for less frequent users such as project managers, software engineers, test engineers and quality assurance managers. This tool has web interface and provides access to users who can perform basic editing and reviewing tasks. Users can browse, search and submit changes to the requirements data. By providing online access of the database it reduces weekly meetings and conferences.

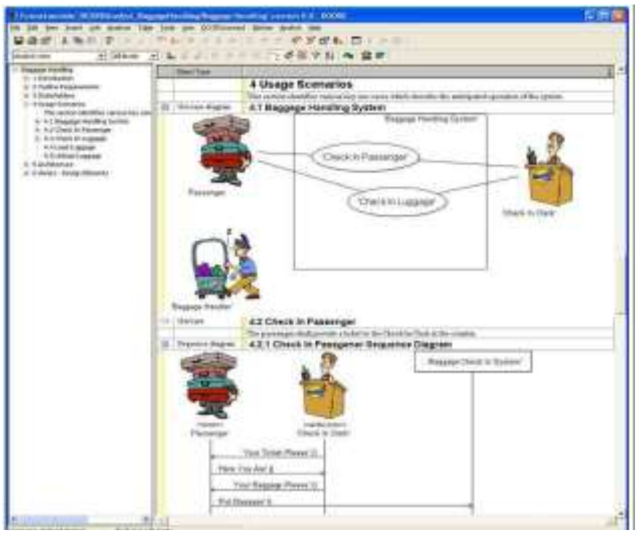


Figure 5.2: DOOR Analyst (reprinted courtesy of [36])

7. CONCLUSION

This paper proposes software metrics in particular requirements metrics to improve the process of managing requirements and quality of the product. Software metrics identify and measures essential factors which effect software development. As collection of requirements is a crucial phase in software development Software Requirements Specification or quality attributes helps in documenting requirements in better way. Writing a perfect SRS is impossible therefore an SRS is considered good which contains as many quality attributes as possible.

Requirements traceability metrics, requirements completeness metrics, requirements volatility metrics and size metrics are used to measure requirements engineering phase of software development lifecycle. Manual measurement is prone to error and time consuming therefore automated measurement tools should be used. Collection of metrics is faster and more reliable with the use of automated requirements tools.

Automated Requirements Tool, Dynamic Object Oriented Requirements Systems, Requirements Use Case Tool and IBM Rational Rose are few automated requirements tools used for collecting, viewing and changing requirements. These tools manage changes and provide traceability metrics. Use of automated tools enhances requirements management but before using any requirements tool its function, uses and limitations must be understood.

REFERENCES

[1]. Alan M. Davis (1993). Software Requirements - Objects, Functions, and States. Prentice Hall, 1993.

[2]. Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap.International Conference Software Engineering, 2000.
[3]. Rita J. Costello and Dar-Biau Liu. Metrics for requirementsengineering. Journal of Systems and Software, 1995.
[4]. WilliamM.Wilson, Linda H. Rosenberg and Lawrence E.Hyatt. Automated quality analysis of natural language requirement specifications. Technical report, Software Assurance Technology Center, 1996.
[5]. Pressman R.S.Software Engineering - A Practitioner’s Approach, McGraw-Hill, 2005.
[6]. M.U.Bokhari and Shams T.Siddiqui. A Comparative study of software requirements tools for secure software Development. BVICAM’S International Journal of IT(BIJIT), 2010.
[7]. Software Metrics SEI Curriculum Module SEI-CM-12-1.1, Carnegie Mellon University Software Institute, Dec 1988.
[8]. Scott Overmyer, Alan Davis, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid,Glen Ledeboer, Patricia Reynolds, Pradip Sitaram, Anh Ta, and Mary Theofanos. The goal question metric approach. Software Metrics Symposium, 1993.
[9]. Rajat R.Sud and James D.Arthur. Requirement Management Tools-A Qualitative Assessment.
[10]. Reichenbach, Bruce R (2001). *Introduction to Critical Thinking*. Boston: McGraw Hill, page 25.
[11]. C.R.Ramakrishnan and J.Rebof(Eds.):TACAS 2008, LNCS $963, pp.463-466, 2008.@ Springer-Verlag Berlin Heidelberg 2008.
[12]. A. Cockburn. Writing Effective Use Cases. Addison-Wesley, 2001.
[13]. Technology Briefing Report System Modeling, 1996.
[14]. James R.McCoy. requirement use case tool.software assurance technology centre. NASA.
[15]. Zohra Bellahsene, Dilip Patel, and Colette Rolland. Object-oriented information systems. In OOIS, pages 409–421, 2002.
[16]. Zave P. and Jackson M. (1997) Four Dark Corners of Requirements Engineering, ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 1, pp. 1-30.
[17]. Dr. Paul Dorsey, Top 10 reasons why systems projects fail.
[18]. Robert B. Grady, “Practical Software Metrics for project Management and Process Improvement”, Prentice-Hall, Englewood Cliffs, 1992.
[19]. Bray I. K. (2002) An Introduction to Requirements Engineering, Addison Wesley.
[20]. Anton A. I (1996) Goal-Based Requirements Analysis.Proceedings of the 2nd International Conference on RE (ICRE '96), 1996.
[21]. BOOTSTRAP (1993) BOOTSTRAP Team, BOOTSTRAP: Europe’s Assessment Method, IEEE Software.
[22]. Wang Y. and King G. (2000) Software Engineering Processes:Principles and Applications, CRC Press.
[23]. Kotonya G. and Sommerville I. (1998) Requirements Engineering, Processes and Techniques, John Wiley & Sons Ltd, 1998.
[24]. Fagan M. E. (1999) Design and Code Inspections To Reduce Errors In Program Development, IBM-Systems-Journal, Vol.38, No., 2-3, pp. 258-287.
[25]. CMMI Product Team (2002) Capability Maturity Model® Integration (CMMISM), Version 1.1, CMMISM for Systems Engineering, Software Engineering, Integrated Product and Process Development, and SupplierSourcing, CMMISE/SW/IPPD/SS, V1.1), Staged Representation, CMU/SEI-2002-TR-012, ESC-TR-2002-012, March 2002.
[26]. Darimont R., Delor E., Massonet P. and Lamsweerde A. V.(1998) GRAIL/KAOS: An Environment for Goal-Driven Requirements Analysis, Integration and Layout, Proceedings ICSE'98 - 20th International Conference on Software Engineering, IEEE-ACM, Kyoto, April 1998, pp. 140.
[27]. Lamsweerde A. V. (2001) Goal-Oriented Requirements Engineering: A Guided Tour, Proceedings RE’01, 5th IEEE International Symposium on Requirements Engineering, Toronto, August 2001, 249-263.
[28]. Murata T. (1989) Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, Vol. 77, No. 4, pp. 541-80.
[29]. SEI (1991) Software Engineering Institute Requirements Engineering Project, Requirements Engineering and Analysis Workshop Proceedings, Technical Report CMU/SEI-91-TR-30, Software Engineering Institute, Carnegie Me note llon University.
[30]. SPICE (1998) Information Technology – Software Process Assessment, Geneva, 1998
[31]. Spivey J. M. (1998) Understanding Z: A Specification Language and Its Formal Semantics, Cambridge University Press.
[32]. ISO9001 (1994) Quality Systems – Model for Quality Assurance in Design, Development, Production, Installation, and Servicing, International Organization for Standardization, Geneva.
[33]. http://en.wikipedia.org/wiki/Requirements_analysis
[34]. Gerald Kotonya and Ian Sommerville. Requirements Engineering. John Wiley and Sons, 2004.
[35]. Karl E. Wiegers(2003). Software Requirements. Microsoft Press.
[36]. Telelogic. Requirements Engineering Tools. http://www.telelogic.com/corp//-products/doors/doorsanalyst/index.cfm(visited 2005-10-24).

[37]. IBM. The IBM Rational RequisitePro. ftp://ftp.software.ibm.com/software//-rational/web/datasheets/version6/reqpro.pdf (visited 2005-10-24).

[38]. http://scarpedia.com/general/requirement-analysis-and-negotiation.