WRITING QUALITY REQUIREMENTS (SRS): AN APPROACH TO MANAGE REQUIREMENTS VOLATILITY

RANJANA RAJNISH
Amity Institute of Information Technology, Amity University,
Lucknow, Uttar Pradesh, India, ranjanavyas@rediffmail.com

PROF. (DR.) HARSH DEV,
Department of Computer Applications, IISE,
Lucknow, Uttar Pradesh, India, doctorharshdev@rediffmail.com

RAJNISH VYAS,
Insta Conn Services & Sales,
Lucknow , Uttar Pradesh, India, instaconn@rediffmail.com

Abstract

Quality requirements, that form a major sub-category of requirements, define a broad set system-wide attributes such as security, performance, usability and scalability. Unfortunately, many organizations pay less attention to quality requirements & assume that the necessary qualities are implicitly understood & will naturally emerge as the product is developed.

In this paper, we would try to address important questions related to role of quality requirements in the software development lifecycle, and techniques for managing them effectively. Also, we would find out as to what happens when requirements are not managed? and suggest some practices to write quality requirements. The purpose is to make the community specifying the requirements understand requirements defects that have been made in the past. Hopefully, it also helps to avoid similar mistakes in the future.

Keywords: *Requirement Engineering, Characteristics of Quality requirements, Requirements Volatility, SRS, Requirement guidelines*

1. Introduction

It looks like your project is off to a good start. The team got some customers involved in the requirements elicitation stage and you actually wrote a software requirement specification. The specification was kind of big, but the customers signed off on it so it must be okay. Now you are designing one of the features and you've found some problems with the requirements. You can interpret requirement 15 a couple of different ways. Requirement 9 states precisely the opposite of requirement 21; which should you believes? Requirement 24 is so vague that you haven't got a clue what it means. You just had an hour-long discussion with two other developers about requirement 30 because all three of you thought it meant something different. And the only customer who can clarify these points won't return your calls. You're forced to guess at what many of the requirements mean, and you can expect to do a lot of rework if you guess wrong.

Many software requirements specifications (SRS) are filled with badly written requirements. Because the quality of any product depends on the quality of the raw materials fed into it, poor requirements cannot lead to excellent software. Sadly, few software developers have been educated about how to elicit, analyze, document, and verify the quality of requirements. There aren't many examples of good requirements available to learn from, partly because few projects have good ones to share, and partly because few companies are willing to place their product specifications in the public domain.

While requirements engineering and requirements tools have become widely adopted, the number of software project failures attributed to poor requirements management remains high. Requirements management helps managers prevent software project failures. The importance of requirements engineering has been gradually recognized by the software industry but still it is hard to say that RE research results are widely in use. One of the major obstacles must be the shortage of evidence that requirements quality really affects outcomes of software project

A report of a survey conducted by the Standish Group in 1994 [9] is repeatedly cited, when arguing about software project success and failure. Although some concerns are raised whether the report actually represents reality and is well supported by scientific facts, the impact of the report was really strong, because the reported failure rate of software projects was staggering 70% or more. But another reason for its wide circulation might be that the failure rate counted by the number of cost overrun and time overrun projects was quite straightforward and easily accepted by the management of software development organizations. The Standish report also shows some data on project success and failure factors. They are based on a survey of IT executive managers asking their opinions why projects succeeded or failed. ***"Clear statement of requirements" was the third in rank (13.0%) of project success factors responded and "incomplete requirements" was the first in rank (13.1 %).*** These numbers appear to imply relation between requirements quality and project success/failure but the data only show subjective perception of managers. Moreover, the quality of requirements is not analyzed in detail. The aim of our study is to focus on the relation between requirements quality and project outcomes.

Before, we look at the attributes for quality requirement, it is essential to understand many other points like what is a typical requirement process? What are characteristics for requirement quality? What happens when requirements are not managed? Some poor quality requirement scenarios, How requirements quality affect the project success? and finally some proposed practices to write quality requirements.

2. A Typical Requirements Process

A requirements process first describes the purpose of a software product, and then refines that purpose into greater detail. Commonly, requirements are set out in one or more requirements documents, generically called work products. While there are special languages designed to express requirements, the natural, English language is still the most widely used. Diagrams, such as the use case diagram, are becoming increasingly popular as a means to convey requirements more efficiently.

A candidate requirement must

- be observable from a point of view that is external to the system
- and, should satisfy some need of the potential customer or other stakeholder

For small and medium-sized projects, typical requirement engineering activities define the high-level need for the system, refine the need into specific features and functions, and build lower level requirements for major components or functional areas. Requirements are also refined into software design requirements.

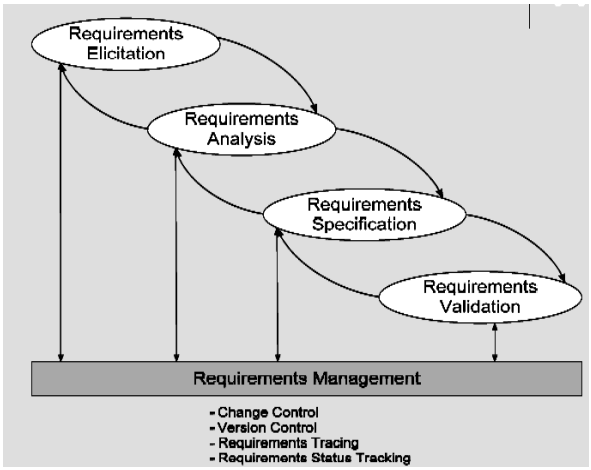

Fig 1. Requirements Engineering Process Life Cycle
Source: "Understanding the effects of Requirements Volatility", Nurie Nurmuliani, Jun'08

Finally, test plans (which are requirements documents for testing) ensure that the requirements captured in the initial high-level requirements documents are satisfied. Fig 1 depicts the typical flow of requirements through the software

development lifecycle. The figure does not apparently concern for managing the process. In almost all requirements process diagrams, the management of requirements management is missing and only the technical part is shown.

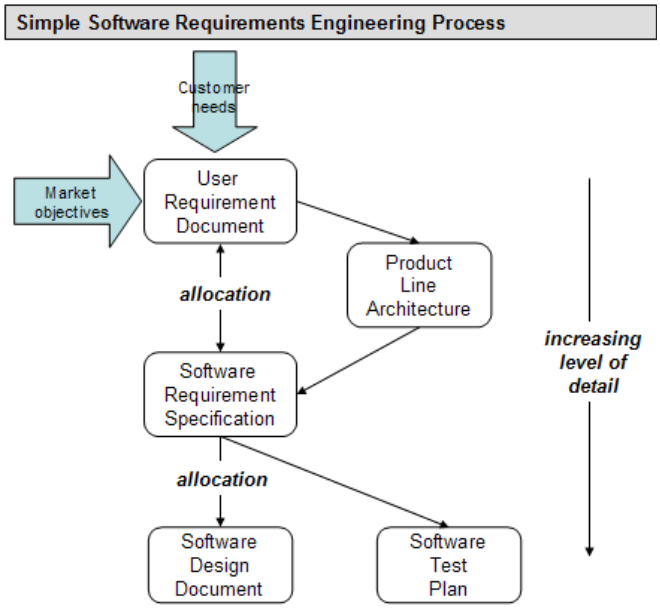

Fig 2. Typical flow of requirements during System Development Life Cycle

3. How Requirement Volatility Effect SDLC?

Although, the initial sets of requirements are well documented, requirements changes will occur during the software development lifecycle as Software development is considered to be a dynamic process where demands for changes seem to be unavoidable. These changes take place while the requirements are elicited, analyzed and validated and after the system has gone into service, simply throughout the software development lifecycle. This change in Requirements during the system development is known as **Requirements Volatility**. These constant changes (addition, deletion, modification) in requirements during the development lifecycle have great impact on the cost, the schedule and the quality of final product. Every phase of software development is effected by requirements volatility. ***Many projects fail due to requirements volatility and some are completed partially***.

Requirements volatility often results in significant growth in requirements size from the time of initial requirements specification to final requirements of the system development. Requirements volatility is an important risk in software project success that can occur in multiple points during the software development process.

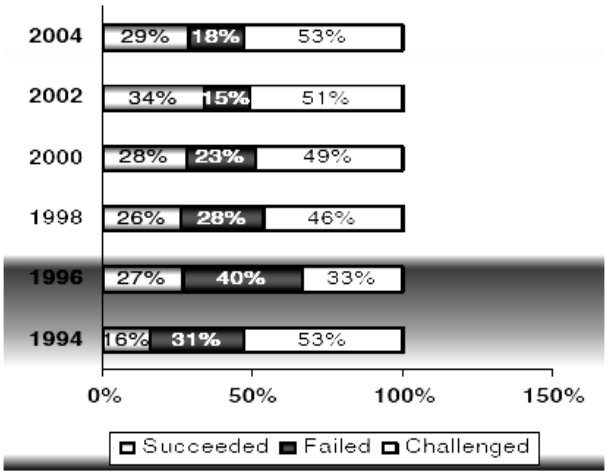

Fig 3. Project Failure Records(Failure Due To Incomplete/Changing Requirements)
Source: The Standish Group Report, 2004

Requirements volatility has a great impact on software development life cycle. The requirements volatility affects software releases and has major impact on project schedule, cost and project performance. The degree of requirements volatility is negatively associated with software project schedule and cost performance. Lamswede conducted a survey, from the data collected from over 8000 projects from 350 companies in USA and revealed that one third of the projects were never completed, and half succeed only partially, i.e. with partial functionalities. Many projects have cost overruns and significant delays.

4. Software Requirement Specifications (SRS)

To the customers, suppliers, and other individuals, a good SRS should provide several specific benefits such as:

a) Establish the basis for agreement between the customers and the suppliers on what the software product is to do.
b) Reduce the development effort.
c) Provide a baseline for validation and verification.
d) Facilitate transfer.
e) Provide a basis for estimating costs and schedules.
f) Serve as a basis for enhancement.

The basic issues that the SRS writer(s) shall address are the following:

a) *Functionality:* What software is supposed to do?
b) *External Interfaces:* How does the software interact with people, the system's hardware, other hardware and other software?
c) *Performance:* What is the speed, availability, response time, recovery time of various software functions etc?
d) *Attributes:* What are the portability, correctness, maintainability, security etc Considerations?
e) *Design constraints imposed on an implementation:* Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

Table of Contents

Fig 4. Prototype SRS Outline

Some of the characteristics that a good-quality requirement should exhibit, that are missing in poorly specified requirements are as below:

•**Correct –** To ensure that SRS correctly reflects the actual needs.

•**Unambiguous –** An SRS is unambiguous if, and only if, every requirement stated has only one interpretation.

•**Complete --** All conditions under which the requirement applies are stated and it expresses a whole idea or statement

•**Consistent –** Refers to internal consistency, and must ensure that it does not conflict with other documents such as system requirement specification.

•**Verifiable –** An SRS is verifiable if, and only if, every requirement stated therein is verifiable, i.e. if there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirements.

•**Traceable –** An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement document.

•**Modifiable—**An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style.

•**Ranked for importance and/or stability—**Each requirement in an SRS must be ranked for importance and/or stability.

5. Some Poor Quality Requirements Scenarios

These descriptions of characteristics of quality requirements are all very fine in the abstract, but what do good requirements really look like? To make these concepts more tangible, let’s do a little practice. Following are few requirements adapted from actual projects. We evaluate each one against the preceding quality criteria to see if it has any problems, and try to rewrite it in a better way. Some suggestions for improvement of each one are mentioned, but you might come up with a different interpretation. Since you and I aren’t the real customers, we can only guess at the actual intent of each requirement.

Table 1: Some poorly stated requirement scenarios

Requirement Specification Scenarios	**Improvement Suggestions**
Example #1: *"The product shall provide status messages at regular intervals not less than every 60 seconds."*	***This requirement is incomplete, ambiguous & not verifiable:*** What are the status messages and how are they supposed to be displayed to the user? The requirement contains several ambiguities. What part of "the product" are we talking about? Is the interval between status messages really supposed to be at least 60 seconds, so showing a new message every 10 years is okay? Perhaps the intent is to have no more than 60 seconds elapse between messages; would 1 millisecond be too short? The word "every" just confuses the issue. As a result of these problems, the requirement is not verifiable. Here is one way we could rewrite the requirement to address those shortcomings: 1. Status Messages. 1.1. The Background Task Manager shall display status messages in a designated area of the user interface at intervals of 60 plus or minus 10 seconds. 1.2. If background task processing is progressing normally, the percentage of the background task processing that has been completed shall be displayed. 1.3. A message shall be displayed when the background task is completed. 1.4. An error message shall be displayed if the background task has stalled." I have split this into multiple requirements as each will require separate test cases and that each should be separately traceable. If several requirements are strung together in a paragraph, it is easy to overlook one during construction or testing.
Example #2: *"The product shall switch between displaying and hiding non-printing characters instantaneously."*	***Requirement is not feasible, incomplete and cannot be verified:*** As Computers cannot do anything instantaneously, so this. It is incomplete because it does not state the conditions that trigger the state switch. Is the software making the change on its own under some conditions, or does the user take some action to stimulate the change? Also, what is the scope of the display change within the document: selected text, the entire document, or something else? There is an ambiguity problem, too. Are "non-printing" characters the same as hidden text, or are they attribute tags or control characters of some kind? As a result of these problems this requirement cannot be verified. A better way to write the requirement can be: "The user shall be able to toggle between displaying and hiding all HTML markup tags in the document being edited with the activation of a specific triggering condition." Now it is clear that the non-printing characters are HTML markup tags. This requirement does not constrain the design because it does not define the triggering condition. When the designer selects an appropriate triggering condition, you can write specific tests to verify correct operation of this toggle.
Example #3: *"The HTML Parser shall produce an HTML markup error report which allows quick resolution of errors when used by HTML novices."*	**Requirement is ambiguous, incomplete & not verifiable:** The word "quick" is ambiguous. The lack of definition of what goes into the error report is a sign of incompleteness. I'm not sure how you would verify this requirement. Find someone who calls herself an HTML novice and see if she can resolve errors quickly enough using the report? Try this instead: "The HTML Parser shall produce an error report that contains

	the line number and text of any HTML errors found in the parsed file and a description of each error found. If no errors are found, the error report shall not be produced." Now we know what needs to go into the error report, but we've left it up to the designer to decide what the report should look like. We have also specified an exception condition: if there aren't any errors, don't generate a report.
Example #4: *"Charge numbers should be validated on-line against the master corporate charge number list, if possible."*	**Requirement is ambiguous, incomplete & not verifiable:** I give up, what does "if possible" mean? If it's technically feasible? If the master charge number list can be accessed on line? Avoid imprecise words like "should." The customer either needs this functionality or he doesn't. Some requirements specifications use keywords like "shall", "will", and "should" as a way to indicate priority. According to me an improved version of this requirement: "The system shall validate the charge number entered against the on-line master corporate charge number list. If the charge number is not found on the list, an error message shall be displayed and the order shall not be accepted."

The difficulty developers will have in understanding the intent of each poorly written requirement is a strong argument for having both developers and customers review requirements documents before they are approved. Detailed inspection of large requirements documents is not fun. The people who have done it, feel it was worth every minute they spent on the inspection. It is much cheaper to fix the defects at that stage than later in the development process or when the customer calls.

6. What Happens When Requirements Are Not Managed

Study after study has identified poor requirements management as a key factor in project failures. There are countless studies spanning the last 35 years that cite poor requirements management as a primary or contributing factor in the failure and even cancellation of software development projects.

The inability to manage requirements leads to projects which do not deliver the functionality that end-users or customers expect or need. Frankly, after decades of failure and with the tools to actually manage requirements, we should have learned by now.

In a July 2005 IEEE article entitled "Why Software Fails -We Waste Billions Of Dollars Each Year on Entirely Preventable Mistakes", Robert Charette lists ***"Badly Defined System Requirements" as one of the primary causes of software project failure.*** He estimates that software failures have cost the US economy as much as $75 billion dollars over the past five years.

In 1994, the Standish Group released The Chaos Study which cited ***"Incomplete requirements(13.1%)" as the number one impairment factor in failed projects.*** The number six factor is "Changing Requirements and Specifications(8.7%)".

7. Good Quality SRS v/s Project Success

Specifying quality requirements such as safety and availability, helps to mitigate risks that stakeholders will not accept in the delivered system because the system is not as safe or available as expected.

In order to deal with quality requirements properly, they should be formulated such that they are measurable. Unfortunately, as there are no easily applicable standard metrics for measuring most qualities, the effort required to quantify and measure quality requirements may exceed the value they add. So the real challenge of specifying quality requirements is that of achieving a proper balance between the cost of specifying them and the value of reducing the acceptance risk.

Observing these guidelines and reviewing the requirements formally and informally, early and often, requirements will provide a better foundation for product construction, system testing, and ultimate customer satisfaction.

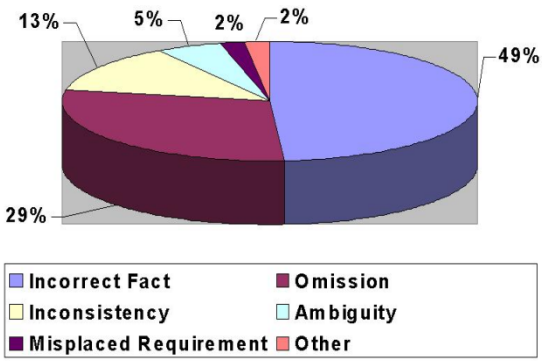

Fig 5. Types of Requirements Errors
(with thanks to Ivy Hooks)

In a study conducted in a large business application software development division of a company, data of requirements specification quality evaluated by software quality assurance teams as well as overall project performance data in terms of cost and time overrun were available. Requirements specification quality data were first converted into a multiple-dimensional space, each dimension corresponding to an item of the recommended structure of software requirements specifications (SRS) defined in IEEE Std. 830-1998. Various statistical analysis techniques were applied over the SRS quality data and project outcomes.

Some interesting relations between requirements quality and project success or failure were found:

1. Data indicate there is relationship between SRS quality and project outcomes. Moreover, a relatively small set of SRS items have strong impact.

2. Descriptions of SRS in normal projects tend to be balanced.

3. SRS descriptions in Section 1, where purpose, overview and general context of SRS are written, are rich in normal projects and poor in overrun projects.

4. When the descriptions of SRS Section 1 are poor while those of functions and product perspective are rich, the project tends to result in a cost overrun, because such characteristics often indicate that the RE phase has been neglected or absorbed in the design phase.

5. Particularly, when references or purpose in Section 1 are well written, the project tends to finish on time.

6. Identifying requirements whose implementation may be delayed can be a good way for preventing cost overrun.

In a July 2005 IEEE article entitled “Why Software Fails -We Waste Billions Of Dollars Each Year on Entirely Preventable Mistakes”, Robert Charette lists ***“Badly Defined System Requirements” as one of the primary causes of software project failure.*** He estimates that software failures have cost the US economy as much as $75 billion dollars over the past five years.

In 1994, the Standish Group released The Chaos Study which cited ***“Incomplete requirements(13.1%)” as the number one impairment factor in failed projects.*** The number six factor is “Changing Requirements and Specifications(8.7%)”.

Thus, my suggestion is that improving the quality of Software Requirements may help in reducing requirements volatility and thus leading to more successful projects.

8. Proposed Practices For Writing Quality Requirements

There is no formulaic way to write excellent requirements. It is largely a matter of experience and learning from the requirements problems you have encountered in the past. Here are a few guidelines to keep in mind as you document software requirements.

- ✓ Keep sentences and paragraphs short. Use the active voice. Use proper grammar, spelling, and punctuation. Use terms consistently and define them in a glossary or data dictionary.
- ✓ To see if a requirement statement is sufficiently well defined, read it from the developer's perspective.
- ✓ Requirement authors often struggle to find the right level of granularity. Avoid long narrative paragraphs that contain multiple requirements. A helpful granularity guideline is to write individually testable requirements. If you can think of a small number of related tests to verify correct implementation of a requirement, it is probably written at the right level of detail. If you envision many different kinds of tests, perhaps several requirements have been lumped together and should be separated.
- ✓ Watch out for multiple requirements that have been aggregated into a single statement. Conjunctions like "and" and "or" in a requirement suggest that several requirements have been combined. Never use "and/or" in a requirement statement.
- ✓ Write requirements at a consistent level of detail throughout the document. I have seen requirements specifications that varied widely in their scope. For example, "A valid color code shall be R for red" and "A valid color code shall be G for green" might be split out as separate requirements, while "The product shall respond to editing directives entered by voice" describes an entire subsystem, not a single functional requirement.
- ✓ Avoid stating requirements redundantly in the SRS. While including the same requirement in multiple places may make the document easier to read, it also makes maintenance of the document more difficult. The multiple instances of the requirement all have to be updated at the same time, lest an inconsistency creep in.

Observing these guidelines and reviewing the requirements formally and informally, early and often, requirements will provide a better foundation for product construction, system testing, and ultimate customer satisfaction.

The classical notion of requirements quality focuses on Adequacy, Unambiguity, Completeness, Consistency, Verifiability, Modifiability and Traceability (IEEE 1993). In practice however, most requirements specifications do not meet these qualities. One could argue that this is only a problem of applying the right methods and processes and that we should improve our requirements processes until they yield the desired qualities. However, a closer look reveals that it is not so simple. The qualities themselves are part of the problem.

The notion of *completeness* leads to waterfall-like process models, where a requirements specification of the complete system has to be produced and base lined prior to any design and implementation activities. However, customers do not always fully know and understand what they want. Systems and requirements evolve. So it is almost impossible to produce and freeze a complete requirements specification.

Unambiguity requires the specification to be as formal as possible. However, in the vast majority of requirements specifications, requirements are stated informally with natural language or at best semi-formally, for example with class models or dataflow models. Thus, unambiguity is very difficult to achieve. The value of *traceability* ranges in practice from irrelevant (many in-house projects) to extremely important (safety-critical projects).

On the other hand, we also do have process- and method-related problems. In many projects, customers are unable to assess the adequacy of the requirements because the way that the requirements are represented does not match the way that customers use a system and think about it. Moreover, when customers do not fully know and understand what they want, the assessment of adequacy becomes even more difficult.

Consequently, we need both a shift in the basic paradigm of requirements quality and a proper adaptation of requirements engineering techniques in order to meet the modified set of qualities. I advocate a requirements quality model that focuses on *adequacy* as the most important quality, views *consistency*, *verifiability* and *modifiability* as next important but deliberately lives with *incompleteness* (that means with partial specifications) and with some *ambiguity*. The weight of *traceability* should be made dependent on the project in hand.

Adopting this new quality paradigm demands requirements engineering techniques that
• describe requirements such that customers can easily understand and validate them
• allow the systematic construction of partial specifications, and
• support the early detection and resolution of ambiguities.

Conclusion

Requirements are the basis of all of the technical work performed on projects. Requirements errors are the largest class of errors typically found in a project (41-56% of errors discovered). Reducing requirements errors is the single most effective action developers can take to improve project outcomes. There is great leverage (and cost savings) in identifying omitted requirements and in finding errors in the requirements stage vs. in later stages of the life-cycle. The cost of rework is typically 45% of projects. Requirements-related activities can reduce rework.

The quality of software development efforts improves when the investment in requirements-related activities is increased from the industry average of 3% to an optimum level of 8-14% of total project costs. We can improve software quality by paying more attention to requirements activities. Requirements are important and provide many opportunities to further strengthen and improve what we are doing. A set of "Effective Requirements Practices" was suggested to be used while writing SRS. It is only by making a few improvements in the actual practices of what we do that we will get better.

Acknowledgment

I wish to express my sincere thanks to my guide for providing his valuable guidance while preparing the paper. I also thank him for reviewing my paper and giving his suggestions for further improvement. I would also like to thank my head of the department for providing me constant encouragement.

References

[1] [Sommerville97] Ian Sommerville and Pete Sawyer: *Requirements Engineering: A Good Practices Guide*, John Wiley & Sons, 1997.
[2] Ian Soumerville, *Software Engineering*, 7th Edition, Pearson Education
[3] D. Zowghi**,** "A Longitudinal Study of Requirements Volatility in Software Development", in the ASMA/SQA Meeting, 2005.
[4] Donald Firesmith: "Prioritizing Requirements", in *Journal of Object Technology*, vol. 3, no. 8, September-October 2004, pp. 35-47. http://www.jot.fm/issues/issue_2004_09/column4
[5] Alan M. Davis, Didar Zowghi, "Good requirements practices are neither necessary nor sufficient", Springer-Verlag London Limited 2004
[6] Donald G. Firesmith: "Common Requirements Problems, Their Negative Consequences, and Industry Best Practices to Help Solve Them", in Journal of Object Technology, vol. 6, no. 1, January-February 2007, pp. 17-33 http://www.jot.fm/issues/issue_2007_01/column2[1]
[7] Ranjana Rajnish: "Requirements Volatility: A Risk in Software Development Process", International Conference on "Emerging Technologies and Applications in Engineering, Technology and Sciences" (ICETAETS-2008), Jan'2008, Saurashtra University, Rajkot, Gujrat, India.
[8] K. E. Emam and N. H. Madhavji. Measuring the success of requirements engineering processes. In Second IEEE International Symposium on Requirements Engineering**,** pages 204-211, 1995.
[9] Standish Group International. The chaos report (1994). http://www.standishgroup.com/sample-research/chaos-1994-l.php, 1994.
[10] R. L. Glass. The standish report: Does it really describe a software crisis? Communications of the ACM,49(8):15-16, Aug. 2006.
[11] IEEE. Recommended practice for software requirements specifications. Technical report, IEEE, 1998. IEEE Std 830-1998.
[12] J. Verner, **K.** Cox, **S.** Bleistein, and N. Cerpa. Requirements engineering and software project success: An industrial survey in Australia and the U.S. 13;225-238, 2005.
[13] D. Beny, D. Damian, A. Finkelstein, D. Gause, R. Hall, E. Simmons, and A. Wassyng. To do or not to do: If the requirements engineering payoff is so good, why aren't more companies doing it? In Proc. 13th International Requirements Engineering Conference (RE105), page 447. IEEE,2005.