A Domain-Specific approach to Verification & Validation of Software Requirements

by

Rehman Chughtai

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Computing Studies

Approved February 2012 by the
Graduate Supervisory Committee:

Arbi Ghazarian, Chair
Ajay Bansal
Bruce Millard

ARIZONA STATE UNIVERSITY

May 2012

ABSTRACT

Gathering and managing software requirements, known as Requirement Engineering (RE), is a significant and basic step during the Software Development Life Cycle (SDLC). Any error or defect during the RE step will propagate to further steps of SDLC and resolving it will be more costly than any defect in other steps. In order to produce better quality software, the requirements have to be free of any defects. Verification and Validation (V&V) of requirements are performed to improve their quality, by performing the V&V process on the Software Requirement Specification (SRS) document.

V&V of the software requirements focused to a specific domain helps in improving quality. A large database of software requirements from software projects of different domains is created. Software requirements from commercial applications are focus of this project; other domains embedded, mobile, E-commerce, etc. can be the focus of future efforts. The V&V is done to inspect the requirements and improve the quality. Inspections are done to detect defects in the requirements and three approaches for inspection of software requirements are discussed; ad-hoc techniques, checklists, and scenario-based techniques. A more systematic domain-specific technique is presented for performing V&V of requirements.

DEDICATION

This thesis is dedicated to my beloved late parents for their unconditional love. Also, it is dedicated to all who believe in power of learning. The will to always learn more and the power you achieve by it help improve your life and knowledge.

ACKNOWLEDGEMENTS

I wish to thank my supervisor, Dr. Arbi Ghazarian, for his support and guidance in not only this project, but my learning and understanding the domain of RE. I also want to thank Dr. Kevin Gary for his guidance.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

1 INTRODUCTION

Comprehension of requirements can be one of the major problems faced in developing large and complex software systems [**1**,**2**]. Quality of the whole software system depends on software requirements as these are compiled at an early stage of development. Requirements Engineering (RE) is the process of developing and managing the requirements. Sommerville has defined RE as

> *"The process of finding out, analyzing, documenting, and checking the services and constraints is called Requirement Engineering (RE)"* [**1**]

In the above definition "services" refer to the services provided by the software system, according to the definition of requirement by Sommerville [**1**]. The activities involved in the RE process and their relationship are illustrated in Figure 1.

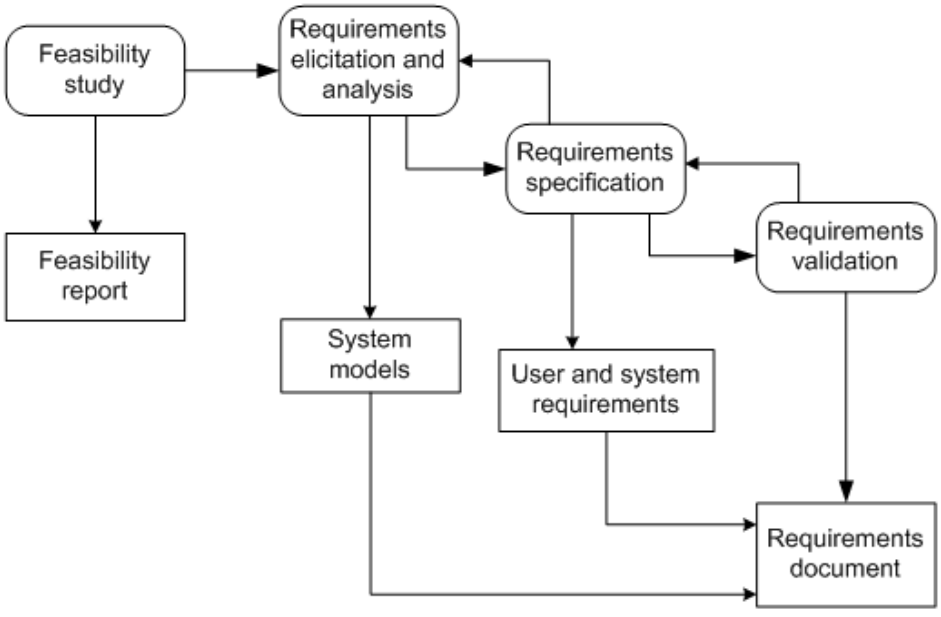

Figure 1 The RE process, adapted from Sommerville **[**1**]**

Another definition of RE provided by Zave [**3**] gives more details.

"Requirements 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 behavior, and to their evolution over time and across software families."

Unlike Sommervile, Zave put emphasis on the role of RE in software engineering and their relationship. Also it shows the importance of RE during the phases of SDLC, as "*...evolution over time and across software families.*" Both Sommerville's [**1**] and Zave's [**3**] definitions show the importance and role of RE in the broader domain of Software and System Engineering. Nuseibeh and Easterbrook have also shown in their work that RE is a multidisciplinary, human-centered process [**4**]. They argue that the tools and techniques used in RE come from different disciplines and RE might need to gain some level of expertise from different domains. Stevens et al. [**5**] have given reasons and arguments in favor of knowledge of system theory, practice and its application are relevant to RE.

Requirements and quality have a relationship, as Crosby [**6**] explicates in his definition of "Quality"; that quality is conformance to the requirements. Thus maintaining better quality implies that all the requirements, of users and other stakeholders, are satisfied [**7**].

Quality maintained during the RE will reflect in all the future phases of software development lifecycle (SDLC). Inspection of requirements helps with

identification and removal of the errors, thus maintaining a better quality in requirements. This results in decreased cost, reduced time for development, and a higher quality end-product. This early detection and removal of defects lowers the development cost of a software project [**8**]. Similarly, Boehm and Basili [**8**] show the importance and cost-effectiveness of early inspections and removal of defects, they maintain that *"Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase."* [**8**]

It is important to note that RE is only limited to early stages of SDLC. A mapping between RE and different stages of development is displayed in the figure 2, which shows the classic V-model of software development. It illustrates testing of a software system against stakeholder needs and other specifications, which are Verification and Validation (V&V), defined and explained in section 2.2, during the SDLC.

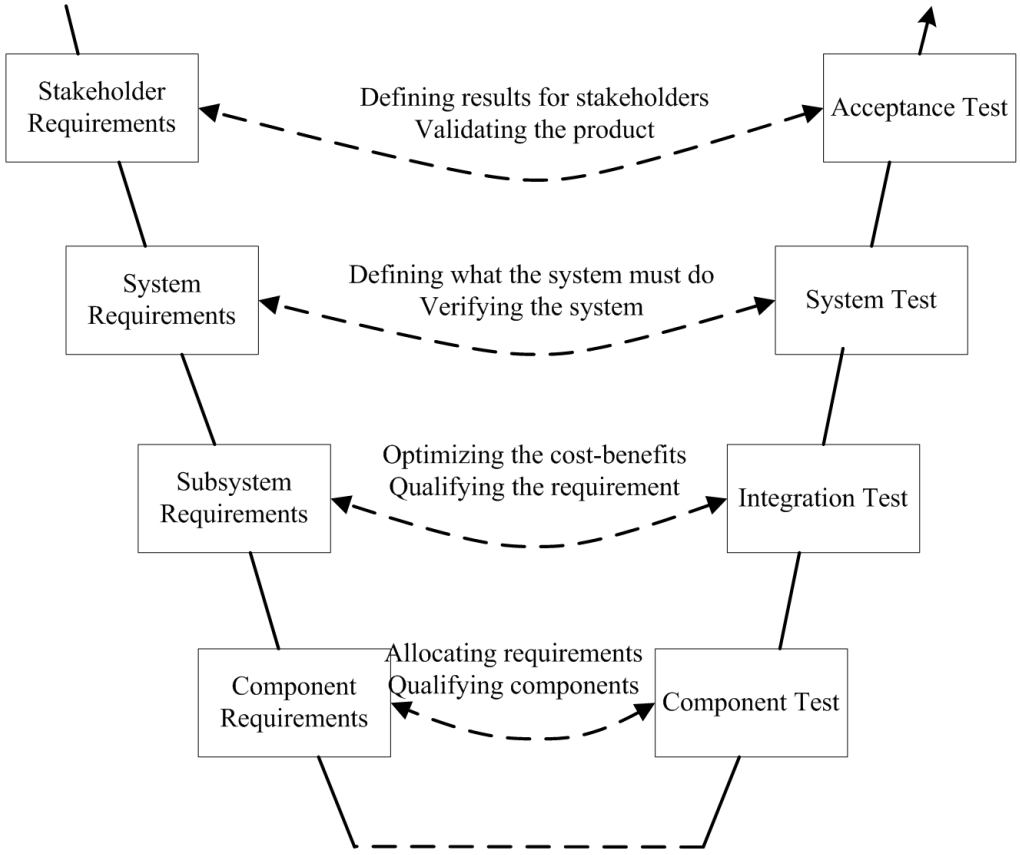

Figure 2 RE in layers of V-model, adapted from Hull et al. [7]

Chapter 2 presents the background and related work; specifically current practices and work done in software requirements inspection. I also go through different inspection techniques currently performed for software and requirements inspection. I discuss, compare and present related work done on the three techniques; ad-hoc techniques, check lists, and scenario-based techniques. Also presented are the types of techniques, i.e., systematic and non-systematic approaches.

In Chapter 3, the discussion continues to the problem of efficient inspection of requirements relative to a domain and how this can affect the quality of not only the requirements but the whole project.

A proposed algorithmic solution for domain-specific verification and validation of requirements is given in chapter 4. Details of the solution are presented in this chapter.

Chapter 5 has the validation of proposed algorithm for the project, by experiments, tests, and their results. This chapter also gives examples of requirements statements, proposed rules for inspecting requirements and results founds after inspection.

In chapter 6, the formalization of the project is presented. The proposed solution is presented in a formal language. Formal language will present the algorithm in a non-ambiguous manner.

Chapter 7 has the conclusion and future work details. I present some ideas for future work and problems based on the results of my project.

There are two appendices in this thesis, which present the supporting data for the proposed solution.

2 BACKGROUND AND RELATED WORK

A successful software system has to satisfy the requirements of its users and environment [**9**,**4**]. One way of conforming to user requirements is by performing an inspection of captured requirements, e.g. software requirements specification (SRS) document. Defective requirements can lead to defects in the final software product, which is not desired by either the end user or developer of the software product. Fixing such defects in later stages of the development process or after the delivery of the software system can be difficult and costly [**1**]. All this makes the software requirements inspection an important process. Inspecting software requirements can also be an important phase in improving quality of software. One of the factors in measuring software quality is the degree to which the delivered software represents the customer requirements [**10**]. Defect-free requirements can be a correct representation of a customer's requirements; inspections help in detecting defects, more specifically, requirement validation helps in ensuring that the requirements represent the customer needs. Cost-effectiveness is another benefit of having requirement documents inspected [**11**]. Requirements inspection helps in identifying and removing defects which prevents defects from spreading into developed software. This means lesser defects the final software product. Also handling defects in the requirements phase is less costly than in later stages or after development of software.

This paper is a review of the current work on different techniques and approaches used for software requirements inspection. This review takes its base from Cheng and Atlee's review paper [**9**] of requirements engineering, specifically the

sections where they review inspections, verification, and validation of software requirements; but this paper provides a more in-depth review of software requirements inspection techniques and related terms which can be helpful in future efforts for developing a more efficient technique. Inspection is often done by a team of inspectors, who review requirement statements in the requirement document in order to identify as many defects as possible.

2.1 Basic terminology

The process of checking software requirements for detecting defects, in order to improve the quality of the resulting software system, is presented in the literature using different terms. These terms include requirement inspection, requirement validation, and requirement review.

Requirement inspection is one of the most commonly used terms in the literature to refer to the process of identifying and removing defects from requirements. The general technique of inspection in software systems was first introduced by Fagan [**12**] in 1976. The original technique was for code and design inspection, but many domains have now adopted it with domain-specific changes, including requirements engineering (RE). Braude [**13**] refers to inspection of software as a process to ensure quality, performed by a team of inspectors. For inspection of software requirements, Porter and Votta [**14**], have referred to this process as the usual method to validate SRS. Runeson et al. [**15**] have recommended inspections as a defect detection technique for requirements inspections.

The inspection process used in many organizations is composed of three steps [**16**,**17**]: (a) defect detection, (b) collection, and (c) repair. Humphrey [**16**] has

termed these three steps as **i.** Preparation, **ii.** Inspection meeting, and **iii.**Repair and report. Each of these steps is briefly explained below:

i. **Preparation**. This step starts with a meeting, where the product to be inspected is introduced; so that every inspector understands it. Here roles are defined and assigned to inspectors, making sure that every inspector knows his or her role and how to perform it.

ii. **Inspection meeting**. After preparation a meeting is held, where findings of each inspector is discussed with the author of requirements and each other. After the defects are noted the responsibility to resolve them is assigned, usually to the author of requirements.

iii. **Repair and report**. In this step the defects collected are repaired and an inspection report is prepared and produced as final output of the inspection process.

Among the three steps, preparation is the most important one because the output of this step, detected defects, affects the total outcome of the inspection process.

Requirements inspection is performed either individually and independently or by a team of inspectors who collaborate in the inspection process.

Quality is related to or can be achieved through some characteristics. For a better quality software requirement, there are some characteristics, which can be found in the literature [**13**,**4**,**18**]. The list of characteristics, which should be checked during an inspection, include: completeness, consistency, feasibility, ambiguity, clarity, preciseness, testability and traceability. Ambiguity is one of the important characteristics; removing any ambiguity from software requirement makes sure

that development follows the correct path. Inspection of informal requirement documents can be performed for detecting ambiguities in requirements before any formal requirement specification is generated [**18**].

Requirement validation is another term used by some authors for the same process of requirements inspection. This is defined as checking the requirement document for a set of characteristics; i.e., consistency, completeness, omissions, ambiguity, and accuracy [**19**,**2**]. Requirements validation is performed to confirm that requirements define customer needs [**1**] . Similar to inspections, the purpose of validation is examining requirements and thus improving the overall quality by checking the requirements against given criteria: i.e. validity checks, consistency, completeness, realism, and verifiability. Verifiability is same as testability; that is a set of tests can be done to show that the final system meets the requirements [**1**].

Another term found in the requirements literature is requirement review. This is a manual process in which, reviewers from both client and contractor organizations physically review the requirements [**1**]. This combination of client and contractor makes the review process more efficient.

The group of reviewers identifies problems in the requirements by analyzing them, discussing the identified problems and agreeing upon some measures to resolve the problems [**19**] . Kotonya and Sommerville [**19**] have presented a complete description of the requirement review process. This is briefly described in Figure 3.

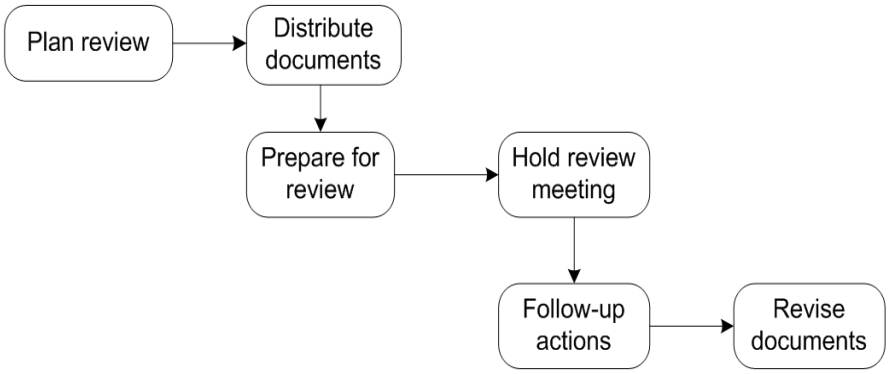

Figure 3 Requirement Review process. Figure adopted from Kotonya and Sommerville **[**19**]**

In Figure 3, the steps shown as blocks are similar to the three steps discussed previously for inspection; i.e. preparation, inspection meeting, and repair and report. As the requirement review process is explained by Kotonya and Sommerville [**19**], the first three blocks, *Plan review*, *Distribute documents*, and *Prepare for review*, perform the same action as the first step of inspection i.e. *preparation*. The next two blocks in Figure 1, *hold review meeting* and *Follow-up actions*, has the same purpose as the second step i.e. *inspection meeting*. The last block of Figure 1, *Revise documents*, performs the *repair and report* step of inspection.

2.2 "Software Verification and Validation" vs. "Requirements Verification and Validation"

Verification and Validation (V&V) are pivotal steps in any project. The concept of verification and validation in the domain of software requirements is slightly different than in software systems and. Verification and validation in software systems and in software requirements is briefly described in subsection 3.1 and 3.2, respectively. These subsections also give the

definitions found in the literature. Boehm [**20**] has presented that the purpose of doing V&V of software requirements is to identify and resolve problems and high-risk issues early in the software life cycle, which saves in costs and time.

2.2.1. Software Verification and Validation

Verification of software is, according to Boehm [**20**], *"The process of determining whether or not the products of a given phase of the software development cycle fulfill the requirements established during the previous phase"*. Pressman [**2**] defines it as *"a set of tasks that ensure that software correctly implements a specific function",* whereas validation is a process which ensures that the software system performs the functions set by stakeholders' requirements. That is to say that validation ensures that the end product, the software system, is according to the requirements set by customer. Boehm [**20**] defines validation as, *"The process of evaluating software at the end of the software development process to ensure compliance with software requirements"*. The definition of software validation by Pressman [**2**]is *"Validation refers to a set of tasks that ensure that the software that has been built is traceable to customer requirements."*

V&V play an important role in software quality, which is clear from the above definitions. According to Boehm [**20**], validation deals with the question "*Are we building the right product?",* whereas verification deals with the question of "*Are we building the product right?"* The purpose of verification is quality, whereas user satisfaction is the goal of validation [**21**].

2.2.2. Verification & Validation of software requirements

The concept of V&V in requirements is a little different than V&V of software systems. Bahill and Henderson [**22**] have defined validation of requirements as making sure that three rules are followed: *"*

1. *the set of requirements is correct, complete, and consistent,*
2. *a model can be created that satisfies the requirements, and*
3. *a real-world solution can be built and tested to prove that it satisfies the requirements."*

From these three points it is clear that validation ensures that requirements are free of any defect and represent the user needs. The first point talks especially about correctness, completeness, and consistency criteria, which can be used as a checklist during the validation of requirements. Bahill and Henderson [**22**] state that *"each requirement must be verified by logical argument, inspection, modeling, simulation, analysis, expert review, test, or demonstration"*. While this definition of requirements verification is in terms of tools for verification, Pfleeger and Atlee [**23**] define verification of software requirements as "*checking requirements specification document corresponds to requirements definition document"*. Here authors have given two different requirement documents, defined as *"requirement definition document that is aimed at business audience such as clients, customers, and users, and a requirement specification document that is aimed at technical audience such as designers, testers, and project managers"* [**23**]. This shows that requirements verification is ensuring that requirements are correctly transformed from the definition of requirement given

by customer to the requirement specification. Requirements verification techniques can be used to show that software specification conforms to its requirements [**9**]. At a broader level, verification of requirements determines that a work product conforms to requirements, which were initially defined [**24**]. Another approach of verification is presented by Jeffords and Heitmeyer [**25**]. They have given a compositional proof strategy for verifying invariant properties of requirements specification. This work uses a Software Cost Reduction (SCR) [**26**] specification of a system as an example. SCR is a set of techniques for designing software systems. They have given two proof rules: a standard incremental proof rule and a compositional proof rule. Application of the compositional rule is useful because it decomposes a large verification problem into smaller problems. Smaller problems can be then solved more efficiently than the larger problem.

It has been shown that requirements errors not found until later stages of Software Development Life Cycle (SDLC) or after implementation of a software system are many times more expensive to fix than if they were found during requirements stage or before requirements stage is complete [**27**]. This shows the importance of performing V&V during the requirements stage. This importance of performing both verification and validation of software requirements is discussed in detail by Hull et al. [**7**]. A vital objective of performing V&V processes is developing confidence that the software system is according to its intended use [**1**]. This shows that the system must be according to the user's

requirements, thus not only V&V is an important process for software system as a whole but also for software requirements too.

2.3 Techniques for defect detection

Requirement inspection techniques described in the software engineering literature can be categorized in three broad categories, namely the ad-hoc methods, checklists, and the scenario based approach.

Table 1 presents a list of different experiments performed for comparing software requirements inspection techniques. In Table 1, except item 5, all experiments compare the three inspection techniques, i.e. ad-hoc, checklists, and scenario-base approaches; whereas the experiment in item 5 was performed only using checklists and scenario-based techniques. A similar table is given by Regnell et al. [**28**], which shows different studies performed by institutions in industry and academia for comparing inspection techniques for software requirements.

ID	Authors	Year	Techniques	Result
1.	Porter and Votta [**14**] *An Experiment to Assess Different Defect Detection Methods For Software Requirements Inspections*	1994	Ad-hoc methods, Checklists, Scenario-based techniques	▪Scenarios improve defect detection rate
2.	Porter, Votta, and Basili. [**29**] *Comparing detection methods for software requirements inspections: a replicated experiment.*	1995.	Ad-hoc methods, Checklists, Scenario-based techniques	▪Scenarios improve defect detection rate
3.	Cheng and Jeffery [**30**] *Comparing Inspection Strategies for Software Requirement Specifications*	1996	Ad-hoc methods, Checklists, Scenario-based techniques	▪Scenarios improve defect detection rate ▪Commercial systems

4.	Fusaro, Lanubile, and Visaggio [**31**] *A replicated experiment to Assess Requirement inspection techniques*	1997	Ad-hoc methods, Checklists, Scenario-based techniques	▪Scenarios do not improve defect detection rate ▪Replication
5.	Sandahl et al. [**32**] *An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections*	1998	Checklists, Scenario-based techniques	▪Only compares checklist and scenario-based techniques ▪Scenarios do not improve defect detection rate
6.	Lanubile and Visaggio [**33**] *Evaluating defect detection techniques for software requirements inspections*	2000	Ad-hoc methods, Checklists, Scenario-based techniques	▪Focus on PBR technique-scenario-based technique ▪Replication

Table 1 Experiments on comparing software requirements inspection techniques

All experiments in Table 1 from item 2 to the last item are replication of experiment performed by Porter and Votta [**14**]. They [**14**] performed the experiment to show that the defect detecting rate is different for detection techniques. They applied each of the three detection techniques on the software requirement document of engineering based embedded systems. Their work was partly supported by National Aeronautics and Space Administration (NASA), so the focus was embedded systems. According to their experiment result scenario-based technique found the most defects and is more helpful in the defect detection process.

Cheng and Jeffery [**30**] performed the experiment to compare the requirements inspection technique for software requirements of commercial systems. Their

focus was on commercial application because it has more data input and output, file manipulation, user queries, and mathematical computation is basic compared to embedded systems.

Fusaro, Lanubile, and Visaggio [**31**] replicated the original experiment with embedded system software requirements. Their results from the replication showthat scenario-based techniques do not improve the defect detection rate. This difference in result can be due to some differences and constraints in the experiment. Firstly, the subjects who performed the reviewers were undergraduate students and most of them had little or no professional expereince. Secondly, native language of reviers was not english and extensive trainnig was required prior to experiment. Another reason can be that one of the SRS used in experiment was of cruise control system used in automobile. This replication was performed in Italy, the cruise system is not very familiar in europe and thus more extensive pre-experiment training was required in this regard.

Sandahl et al. [**32**] performed an extended replication of the experiment done by Porter et al. [**29**]. They only compared Checklists and scenario-based techniques for inspecting software requirements. Also this experiment manipulated three independent variables: detection method, requirements specification, and the order of the inspections. The paper [**32**] also provides details of experiment performed and statistical data from all the repititions of experiment.

Lanubile and Visaggio [**33**] replicated the experiment to compare the techniques. They have focused on Perspective Based Reading (PBR), a systematic scenario-based technique.

Basili et al. [**34**] has a website that is available with title "*Lab Package for the Empirical Investigation of Perspective-Based Reading*". This website has the details of exeperiment performed for comparing different defect detection techniques for software requirements. The requirements used by them was of two embedded systems; Automated Teller Machine (ATM), Parking Garage control system.

Next three sub-sections provide a brief description of each of three requirements inspection technique.

2.3.1. Ad-hoc methods

Ad-hoc techniques are one of the most basic and commonly used techniques by inspectors [**14**,**31**]. In this type of technique, every inspector is assigned with general responsibility of finding defects within the Software Requirements Specification (SRS) document, without any specific guidelines. In ad-hoc detection methods, all inspectors are given the same general responsibility and no formal method or algorithm is used. Instead, the inspectors use their experience and skill in detecting requirement defects, which make this technique a non-systematic approach. Also, the number of defects found and efficiency of this technique is very much based on the experience and skill level of the inspectors.

2.3.2. Checklists

Checklists are another commonly used technique for defect detection [**29**]. Using them to detect faults in a work product, including SRS, can be helpful to reviewers [**2**,**35**] as they may enlist the most common errors, questions

assisting in the detection process or even prioritize the listing. An inspector can use a list of questions to validate each requirement. Kotonya and Sommerville [**19**] have given examples of such a checklist. Here a listing of criteria to check for faults is used. Such lists can prove helpful during the fault detection process.

Table 2 is an example of analysis checklist given by Kotonya and Sommerville [**19**]. This is a list of questions which should be checked for each requirement by the analyst.

Checklist item	Description
Premature design	Does the requirement include premature design or implementation information?
Combined requirements	Does the description of a requirement describe a single requirement or could it be broken down into several different requirements?
Unnecessary requirements	Is the requirement 'gold plating'? That is, is the requirement a cosmetic addition to the system which is not really necessary?
Use of non-standard hardware	Does the requirement mean that non-standard hardware or software must be used? To make this decision, you need to know the computer platform requirements.
Conformance with business goals	Is the requirement consistent with business goals defined in the introduction to the requirement document?
Requirements ambiguity	Is the requirement ambiguous? What are the possible interpretations of requirement? Ambiguity is not always harmful; it gives system designers little degree of freedom. But, in later stages of development it has to be removed.
Requirements realism	According to the technology to develop the system, is the requirement realistic?
Requirements testability	Is the requirement written in such a way that test engineers can develop test to prove that the system meets the given requirement? Simply, Is it testable?

Table 2 Analysis checklist items. Table abopted from Kotonya and Sommerville **[**19**]**

In addition to checklist items mentioned in Table 1, Kotonya and Sommerville [**19**] have also provided some quality attribute that can be used as a review checklist. These quality attributes include understandability, redundancy, completeness, ambiguity, consistency, organization, and traceability. These attributes are applied during the inspection to the requirements documents as a whole instead of applying on each individual requirement. Other examples of checklists can be found in the literature for requirements inspections, e.g. checklist given by Hull et al. [**7**]. Such lists can be used to create a checklist of criteria to identify defects and missing requirements. Porter and Votta [**14**] have also provided a list of questions, which is part of a checklist. They have categorized the defects into three categories: General, Commission, and Omission. Omission means missing functionality, performance, or environment. Commission means insertion of incorrect or extra data or a requirement which is not listed in the correct place. The third category, i.e. General, represents all the other types of defects that are not Commission or Omission. This categorization is based on the work of Schneider, Martin, and Tsai [**36**].

Table 3 gives a list of examples from literature where checklists are used for inspection of software requirements.

An example of a checklist of inspecting for missing functionality is given in Table 4. This checklist is adopted from Porter and Votta's [**29**] experiment.

Missing Functionality Checklist.

▪ Are the described functions sufficient to meet the system objectives? ▪ Are all inputs to a function sufficient to perform the required function? ▪ Are undesired events considered and their required responses specified? ▪ Are the initial and special states considered (e.g., system initiation, abnormal termination)?

Table 3 Missing functionality checklist. Checklist adopted from Porter and Votta **[**29**]**

Table 5 provides a list of inspections performed using checklists that are found in literature.

ID	Authors	Year	Inspection Techniques
1.	Martin and Tsai [**37**] *NFold Inspection: A Requirements Analysis Technique*	1990	Checklist (N-fold)
2.	Lutz [**38**] *"Targeting Safety-Related Errors During Software Requirements Analysis"*	1993	Checklists
3.	Halling et al. [**39**] *"Tailoring a COTS Group Support System for Software Requirements Inspection"*	2001	Checklists

Table 4 Use of checklists in literature

In Table 5, Martin and Tsai used a traditional inspection of requirements using checklist but replicated the experiment with N independent teams. Lutz [**38**] focused on the use of checklists for software requirements inspection of spacecrafts and other safety critical systems and embedded systems.

Although checklists do help in finding out defects, generality of items in the list and the less systematic approach in this technique results in a less number of defects detected; this is shown in experiments found in the literature [**14**,**29**,**30**,**31**].

2.3.3. Scenario-based approach

Jarke et al. [**40**] have defined the term scenario as *"description of a possible set of events that might reasonably take place"*. A scenario, with respect to

requirements inspection, represents a script or procedure that the inspector should follow [**28**]. In the two techniques given previously in section 4.1 and 4.2, ad-hoc technique is non-systematic and general whereas checklists are less systematic and mostly general. A general responsibility means finding as many defects as possible without assigning special role or responsibility to the inspector. An inspector may completely ignore a defect, repeatedly ignore similar defects, or misidentify a statement as a defect. This is because of the non-systematic and general nature of the inspection techniques being used.

Another definition of scenario is given as "*a collection of procedures that operationalize strategies for detecting particular classes of defects*" [**14**]. Requirement inspection scenarios are related to a set of events or procedures relevant to a specific action, actor, or class of defects.

In the scenario-based defect detection approach, a team of inspectors is required to perform only one scenario at a time and to inspect all requirements with coverage of every scenario is ensured by the team. A scenario-based approach not only uses specific responsibilities, but also classifies defects.

Table 6 presents an example of a scenario from Porter and Votta's [**29**] experiment. This experiment was done for software requirements of an embedded system so the scenario talks about terms like precision, response time, and monitored event; a similar scenario can be developed for other domains with domain-specific terms and requirements. Also this scenario was for the perspective based reading, although this looks like another checklist but this scenario is only related to detecting ambiguities in requirements.

Requirement inspector can use this scenario and inspect the requirements accordingly.

Ambiguities or missing functionality scenario.
1. Identify the required precision, response time, etc. for each functional requirement. ▪ Are all required precisions indicated? 2. For each requirement, identify all monitored events. ▪ Does a sequence of events exist for which multiple output values can be computed? ▪ Does a sequence of events exist for which no output value will be computed?

Table 5 Ambiguities or missing functionality scenario. Scenario adopted from Porter and Votta **[**29**]**

Two variants of scenario-based techniques have been proposed: Defect-Based reading (DBR) [**29**] and Perspective-Based Reading (PBR) [**41**].Regnell et al. [**28**] have described Perspective-based reading (PBR) as an inspection technique for requirement document that "… focuses on the points of view of the users of a document". A set of procedures is provided by PBR to inspect software products for defects [**42**].

The defect-based technique concentrates on specific defect classes, while perspective-based focuses on the points of view of the users of a document. In previous related work, there are examples of experiments performed [**14**,**31**,**43**,**44**], where scenarios are used. Work of Porter and Votta [**14**] in 1998 is an early example of scenario-based approach being used for defect detection, many other replications of the experiment [**29**,**31**] with little alteration were performed. Conclusion of all the experiments shows that scenario-based approach is better in defect detection from other two

approaches. Table 7 enlists some of the work from literature for PBR and DBR.

ID	Authors	Year	Techniques
1 .	Basili, V.R. et al. [**41**] *The empirical investigation of perspective-based reading*	1996	Perspective-Based reading (PBR)
2 .	Forrest Shull et al. [**42**] *How Perspective-Based Reading Can Improve Requirements Inspections*	2000	Perspective-Based reading (PBR)
3 .	Björn Regnell et al. [**28**] *Are the Perspectives Really Different__ Further Experimentation on ScenarioBased Reading of Requirements*	2000	Perspective-Based reading (PBR)
4 .	Fusaro et al. [**31**] A Replicated Experiment to Assess Requirements Inspections Techniques	1997	Defect based reading

Table 6 Perspective and Defect based reading

Parnas and Weiss [**45**] argue that higher efficiency can be achieved through more systematic detection approach with selective responsibility assigned to inspectors. Their work on the active design review, where individual reviewers work on a specific purpose using specialized questionnaire [**45**], motivated Porter and Votta's scenario based approach [**14**] for defect detection in requirements. This results in more efficiency, higher rate of defect identification and removal. Mostly this is done using a modification of checklist technique. In this way each inspector is assigned a unique set of responsibilities and guidelines for how to achieve more efficient result. Thus each inspector has a scenario to inspect the requirements.

Although the experiments have shown that scenario-based approaches produce better results in defect detection, there is still debate on which method is better, this is clear from next discussion.

Porter and Votta [**14**] have reported on their experiment-based work on comparison of three defect detection techniques, namely the ad-hoc techniques, checklist, and the scenario-based techniques. From the results of their experiment, and replicated experiments by Porter, Votta and Basili [**29**], and Cheng and Jeffery [**30**], the scenario-based technique has shown to detect more defects than other non-systematic approaches. However, in contrast to these studies Fusaro et al. [**31**] have also replicated the experiment, but they argue that using the scenarios based technique did not result in significant improvement in defect detection..

2.4 Conclusion

Mostly, software code has been the focus of inspection, but literature and experiences in the domain of software requirements implies that inspections should be carried out in earlier stages of the software development life cycle (SDLC) [**30**]. Performing inspections of software requirements are very helpful in improving the quality of not only the requirements but also the complete software system. Performing inspection of software requirements help in identifying and resolving problems early in SDLC [**20**]. This makes the process of handling defects easier and also requirements defects are more expensive to fix later in SDLC [**4**,**27**]. Therefore requirements inspection can result in reduced development costs.

Non-systematic techniques are commonly used for inspecting software requirements. Software requirements inspection literature has shown that such techniques are less efficient. Thus a systematic technique for software

requirements inspection needs to be developed. A technique which is based on an algorithm for inspecting software requirements and detecting defects can be developed and defect detection rate of this new technique can be compared with current techniques.

Also the commonly used techniques are generally used in software projects of every domain. *Domain-specific* is mentioned as a research strategy by Cheng and Atlee [**9**], so a change in focus from generic to domain-specific can potentially yield significant improvements in the field of requirements inspection.

3 PROBLEM STATEMENT

Given a specific domain, like software for mobile device, commercial software, E-commerce, etc., how to efficiently perform V&V of the software requirements i.e. Domain-specific Verification and Validation (V&V) of software requirements. As shown in Chapter 2 and many works in the literature [**1**,**22**,**24**,**11**] , V&V of software requirements, but the literature also shows that V&V are performed in a systematic manner resulting in in-efficiency.

Incomplete and defective software requirements are one of principal basis of software project failure [**24**]. Defect removal, handling incompleteness, and improving quality by handling other criteria is done by performing Verification and Validation (V&V). Inspection is the most common technique for reviewing software requirements. Experiments and other work on software requirement inspection have been performed, but a Domain-specific approach to do Verification and Validation (V&V) is not under much focus of researchers.

Definition of V&V, its importance, and application of this in the software requirements are discussed previously in section 2.2 and its subsections. One of the problems, mentioned in the sub section 2.5.1, is absence or improper V&V of software requirements. Some of work available in this field is general and non-systematic inspection performed on software requirements.

Validation of requirements is performed by checking the requirement statements in the SRS document for any incompleteness, inconsistency, ambiguities and making sure that they follow a quality standard [**11**]. Verification

is a process to ensure that each phase in SDLC accomplishes the requirements set by last stage [**20**]. This can be applied to software requirements for performing the verification. This way V&V of software requirements can be performed.

Performing V&V with precise information of a specific domain can be more efficient in defect detection and removal. Developing of domain-specific strategies for a project can also help in future projects of same domain. Also the domain-specific information can help in developing better inspection technique; like developing checklist criteria.

Inspections are the process of choice used for V&V of software requirements. Section 2.3 and its subsections give three techniques for defect detection techniques found in the literature. As found by many researchers that the most common techniques use for inspection i.e. Ad-hoc and checklist are not very efficient [**14**,**46**,**30**,**31**].

Figure 5 shows a layered view of General inspections, software inspections, software requirements inspections (applied to every domain without any particular consideration to a specific domain, also most commonly used techniques are non-systematic), then there are software requirements inspection specific for a domain with a systematic approach for inspection.

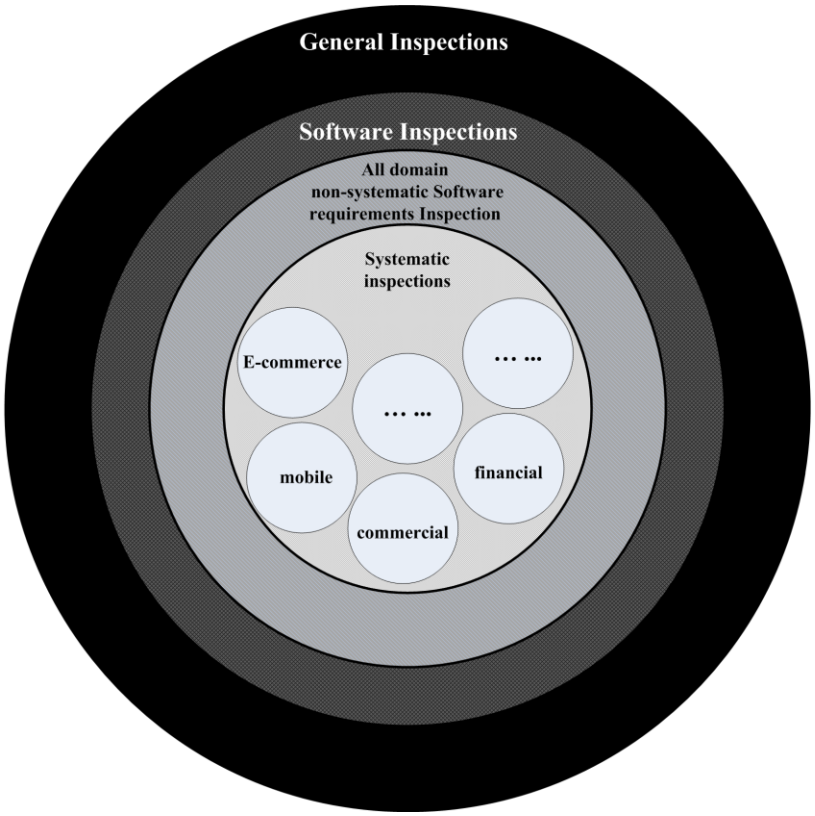

Figure 4 Inspections and Domains

4 PROPOSED SOLUTION

The vital importance of requirements inspection and then performing Verification and Validation (V&V) process is obvious from chapter 2. Also from the literature it is shown that scenario-based techniques, which are systematic or algorithmic in nature, are more efficient [**14**,**46**,**30**,**31**].This shows that an algorithmic technique for inspection yields better results in defect identification. Literature also shows the importance of focusing on a specific domain [**47**].

After studying the literature, an algorithmic approach for performing validation of software requirement is presented, with domain-specific focus.

4.1 Process of solution

The process of my proposed solution for domain-specific validation of software requirements is presented here and the in the sub-sections.

4.1.1 **Introduction**

After studying the literature, it is obvious that most of the inspection techniques are non-systematic and a systematic way to performing software requirement inspection is more efficient, improves quality, and reduces cost. This importance and benefits of a systematic approach are shown by experiments performed by researchers [**14**,**46**,**31**] to compare systematic and non-systematic approaches. One of the benefits of efficiently finding defects is reduction of cost. Boehm and Basili [**8**] have argued that defect removal during the early stages of requirements can be 100 times less expensive than finding defects after delivery.

An algorithmic way of doing inspection of SRS document is presented, which outputs a report of potential defects in SRS. This report is then used to create a checklist; this is a checklist of possible defects based on an algorithm, so it is a systematic method for inspecting software requirements. My proposed algorithm for performing V&V consists of three steps, briefly presented in the figure 6.

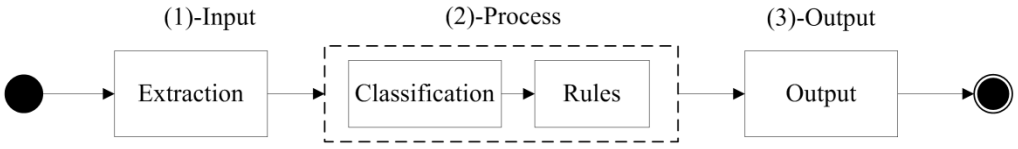

Figure 5 Proposed solution

Next sub-section describes my algorithm in more detail.

4.2 Proposed algorithm

The steps of proposed algorithm are given in figure 7.

Step 1: Extraction.

Step 2: a: Classification

b: Rules

Step 3: Output

Figure 6 Proposed algorithm

These steps are explained in following sub-sections.

4.2.1. Extraction

This is the first, input, step of the process. Here the requirement statements are pulled out of SRS document and input into a database of requirement statement, simply reffered to as *the database* onwards in this thesis. In the extraction we made sure that each single entity of the requirement statement table in the

database is an atomic requirement. Atomic requirement and atomicity is defined by many authors [**7**,**48**,**49**]. Hull et al. [**7**] have defined atomic as *"each statement carries a single traceable element"* [**7**]. Salzer [**49**] has defined atomic requirement as *"indivisible well-formed requirements that enable control over software design, test planning, and work management with an ease and accuracy not previously attainable"* [**49**]. Each requirement statement is manually checked with the definitions of the term atomic, atomic requirement and atomicity. If a requirement statement is non-atomic and is to be broken into two or more than two atomic requirement statements. It is made sure that no redundant requirement or words in statements are added. Also it is to be made sure that while breaking a non-atomic requirement, meaning of the original requirement is not destroyed.

4.2.2. Process

The second step, Process, is composed of two sub steps: Classification and Rules.

4.2.2.1 Classification

First the requirement statements are classified into different requirement types; termed as Classification. This is done according to a classification scheme; after manually studying each statement, each requirement statement is classified according to the action performed in it. For example if a statement is like "user inputs name and password." Then it is clear that an input to the system is done here, so such requirement is of type Data input. The requirement types we found in our project are in appendix A; where Table 14 gives the requirement types and their description. This classification and descriptions can be used in further

studies and classification of requirements. If the action in the statement is not one of the types in table then a new type is defined. For example, in the study while extraction of an SRS of web project many requirement statements were encountered which defined user interface, which were not in my current requirement types at that time. One example statement is *"Pricing is shown on each day on the widget screen below the calendar"*, a new type **User Interface** is assigned to this and new statement with same action.

4.2.2.2 Rules

Second sub-step in the Process is development of rules. The database of software requirements was studied, and a pattern of recurring relationship between requirement types was found. Also some pattern is obvious from a software engineering perspective. One example can be that any data input from user must be validated for input requirements, which can be string format, alpha-numeric character check, or many others. For example, after studying the *database* developed from requirements in SRS document of commercial projects, repeating patterns in it, and missing parts in pattern we have developed a set of rules; these rules can be an example for future work on other domains and development of rules for that domain. Also these rules are developed for one domain and might not applicable in other domains; but some similar rules might come out. Table 7 gives the set of rules that we discovered, for a project related to domain of *commercial*, by studying the *database*. These are 11 rules relevant to the requirements database that were studied.

ID	Rule
1.	For every item of type **Data Input**, there exists at least one item of type **Data Validation**
2.	For every item of type **Data Input**, there exists at least one item of type **Data Persistence**
3.	For every item of type **Data Validation**, there exists at least one item of type **Data Output**
4.	For every item of type **Event trigger**, there exists at least one item of type **other requirement**
5.	For every requirement of type **Data input**, all the input data items for the requirement should be explicitly described
6.	For every requirement of type **Data output**, all the output data items for the requirement should be explicitly described
7.	For every **use-case/feature** There exists at least one requirement of type **data validation** requirement.
8.	For every **use-case/feature** There exists at least one requirement of type **data input** requirement.
9.	For every **use-case/feature** There exists at least one requirement of type **data output** requirement.
10.	For every **use-case/feature** There exists at least one requirement of type **Business Logic** requirement.
11.	For every **use-case/feature** There exists at least one requirement of type **data persistence** requirement.

Table 7 Rules for inspection of software requirements

4.2.3. Output

This is the final step of the algorithm. In this step, after going through the previous steps, a report mentioning possible defects in the SRS is generated as output of the algorithm. The requirement statements are inspected while considering the rules; rules which are generated in second sub-step of classification. With this inspection defects are detected, and these defects are produced as a report.

With this output report now a checklist of defects can be created. Inspector can use this checklist for inspection, but this is not a generic checklist, which is generally applicable on any software requirements document. This is a

systematically generated checklist, based on an algorithm, and specific to the domain of current project.

4.3 Domain-specific

During the three steps, especially during the rules development step, requirements related to a specific domain were focused. This helps in developing rules for that specific domain, and thus finding defects related to that domain. The benefits being domain-specific are previously discussed in chapter 2.

4.4 Conclusion

This algorithmic domain-specific validation of software requirements can be performed on different domains. For any given domain some of the rules may be different, but the output will be specific to that domain and the checklist developed with it will be helpful in finding defects related to that domain.

5 VALIDATION

This chapter gives the validation of proposed solution, given in chapter 4.

For the purpose of validation we use a database of requirements. This database is populated with requirement statements from SRS documents of real life projects. Following sub-sections describe how the three steps of algorithm are applied; three steps are extraction, classification, and output of the algorithm presented in chapter 4.

5.1 Extraction and requirement statements database

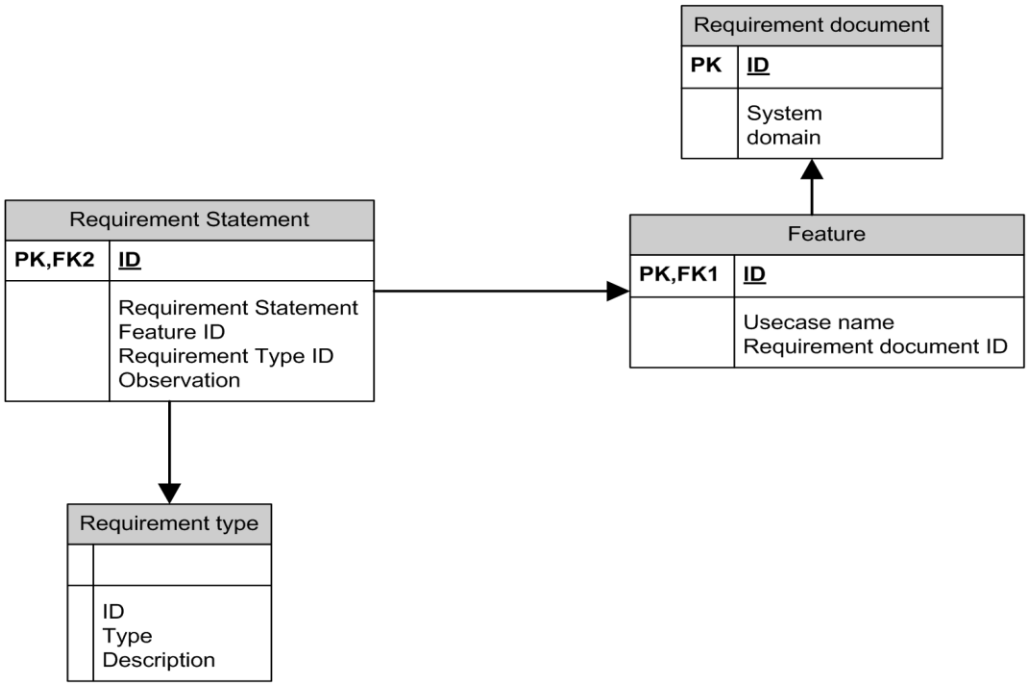

Figure 7 Database of requirements

We used SRS documents of projects from industry; for confidentiality we cannot disclose the name of projects or companies. In the first step of extraction we extracted requirements statements from requirement documents and populated a

database of requirements. Figure 8 shows the developed database; with tables and relationship among them

5.2 Classification of requirements

We studied the requirements in database and as step 2.1 of proposed algorithm classified the requirements. The classified requirements types for selected database are described in Table 12 in appendix A. In appendix A, there is a requirement type called *High Level Requirement*; this is not a requirement type but those requirement statements which were not atomic and not properly defined in SRS were assigned this type.

5.3 Application of Rules and output

We studied requirement statements from real life projects. For our project the rules, discussed previously in section 4.2.2.2 and rules given in Table 7 were applied to these requirements. Following is an example of requirement statements from SRS and how a rule is violated, thus the output of inspection with the rule in consideration is detection of a defect. This example uses rule 1 of Table 7.

> Rule1: "For every item of type **Data Input**, there exists at least one item of type **Data Validation**"
>
> *Req.*: "user enters email address in the page."
>
> Assuming that the related **data validation** is not specified in the requirements document, then the following output is resulted by applying rule 1.

Output: This data input requirement is missing the related data validation requirement.

Problem: Missing requirement

Rule 1, of our discovered rules, says that every data inputted into the system must be validated. In this example requirement 1 (*Req.*) is a data input, where the user inputs the email address. After manually inspecting the database of requirements, there is no ***data validation*** requirement which validates the input email address. This violation of rule 1 shows the defect of a missing requirement, in this case missing requirement of type data validation. That is according to Rule1 a data inputted to system has to be validated in a requirement statement of type data validation, but there is no validation found for this input in the example. This defect is recorded and presented in a meeting with other inspectors and author of requirements, so that the defects are discussed for further processing and are removed or corrected.

In similar way, as above the requirements are checked with rule 1, all the requirements were subjected to all the rules in Table 7. Following table shows examples for next 10 rules. In the examples in Table 8, whenever there are two requirement statements mentioned they are given arbitrary numbers. The numbering *R1* and *R2* do not mean that the two requirements are consecutive

Rule ID	Example
2.	*Rule2:* "For every item of type **Data Input**, there exists at least one

Rule ID	Example
	item of type **Data Persistence**" ***Req. statement****:* "User should input first name in the registration page."
	Assumption: the related data persistence is not specified. *Problem:* Missing requirement *Output:* For this **data input** requirement there is no **data persistence** requirement. The data input requirement inputs a data item into the page which is not stored into or communicated to database that is no data persistence. This shows a missing requirement of type data persistence.
3.	*Rule3:* "For every item of type **Data Validation**, there exists at least one item of type **Data Output**" ***Req. statement****:* "The system verifies that all fields of registration page are filled."
	Assumption: the related data output is not specified. *Output:* For the given example data validation requirement there is not related data output in the requirement document. Thus according to rule 3, this validation statement is missing an output requirement statement. *Problem:* Missing requirement
4.	*Rule4: "For every item of type **Event trigger**, there exists at least one*

Rule ID	Example
	*item of type **other requirement**"* ***Req. statement**:* "The user can click on "Content Management" on the page."
	Assumption: there is no related requirement specified. *Output:* The requirement document has no other requirement statement which describes any action that happens related to this event trigger. There is no related requirement to this trigger. This shows a missing requirement. *Problem:* Missing requirement
5.	*Rule5: "For every requirement of type **data input**, all the data items for the requirement should be explicitly described"* ***Req. statement**:* "The user enters his login credentials into the system."
	Assumption: the related data input items are not explicitly specified. *Output:* the requirement statement should explicitly define the login credentials, like credential can be username and password. So this is a vague and incomplete requirement statement. *Problem:* Incomplete requirement
6.	*Rule 6: "For every requirement of type **Data output**, all the output data items for the requirement should be explicitly described"* ***Req. statement**:* "A menu is displayed on the page with several options."

Rule ID	Example
	Assumption: the related data output items are not explicitly specified. *Output:* the requirement statement should explicitly define that what several options are? So this is a vague and incomplete requirement statement. *Problem:* Incomplete requirement
7.	*Rule6: "For every **use-case/feature** There exists at least one item of type **data validation** requirement."* *Use case ID:* "42"
	Assumption: there is no data validation requirement specified in this usecase. *Problem:* Incomplete use case *Output:* There is no requirement of type data validation in this use case.
8.	*Rule7: "For every **use-case/feature** There exists at least one item of type **data input** requirement."* *Use case ID:* "43"
	Assumption: there is no data input requirement specified in this usecase. *Problem:* Incomplete use case *Output:* There is no requirement of type data input in this use case.
9.	*Rule8: "For every **use-case/feature** There exists at least one item of*

Rule ID	Example
	*type **data output** requirement."* *Use case ID:* "44"
	Assumption: there is no data output requirement specified in this usecase. *Problem:* Incomplete use case *Output:* There is no requirement of type data output in this use case.
10.	*Rule9: "For every **use-case/feature** There exists at least one item of type **Business Logic** requirement."* *Use case ID:* "51"
	Assumption: there is no business logic requirement specified in this usecase. *Problem:* Incomplete use case *Output:* There is no requirement of type business logic in this use case.
11.	*Rule10: "For every **use-case/feature** There exists at least one item of type **Data persistence** requirement."* *Use case ID:* "50"
	Assumption: there is no data persistence requirement specified in this usecase. *Problem:* Incomplete use case *Output:* There is no requirement of type data persistence in this use case.

Table 8 Rules and example

Another example of defective requirement statement is given below. There are some defects in the requirement statement and also some violation of rules.

***Req.**: The user should be able to view item details that have been added previously (stored in local database).*

First problem with above requirement statement (R6) is that it is the only requirement in the use case, given in the SRS. This violates the last five rules of Table 7; which require that every use case must have at least one data input, data output, data validation, business rule, and data persistence. Another problem with this statement is that it is not atomic and can be broken into distinct atomic requirements.

Each rule applied to relevant requirement type and its result is shown in Table 16 in Appendix B; which shows all the statistics of the inspection.

For example in all of the requirements inspected by the experiment, the total number of **data input** items in system 1 are 11. Among the 11 data inputs there are only 4 data inputs which have relevant data validation requirements, and 7 data inputs were missing data validations. From this precision of defect detection is calculated with following formula

$$\text{DDR} = \frac{\text{Total number of Defects found by rule R}}{\text{Total number of cases where rule is applied}}$$

$\therefore$ DDR= Defect Detection Rate

In the above formula, precision is in terms of number of defects found by a rule. For this example percentage of defects found by rule 1 in system 1 is as follow

$$DDR_{11} = \frac{7}{11} = 63.63\%$$

The first number 1 as subscript in DDR_{11} denotes defect detection rate calculated by rule 1 and second is for system 1. In the above calculation $\frac{7}{11}$ shows that out of total 11 cases in requirement statements subjected to this rule, 7 defects were found in system 1. Here it is important mention the found false-positives. Table 9 shows the detected defects after inspecting the requirements with rules in Table 7. This table shows the defects found in each system, after applying the rule. First two columns in the table give the rule id and description of rule. Next four columns are divided into two rows; each row gives the information relative to a system. The third column in Table 9 gives the total number of rule application in requirements. Precision of each rule application is also calculated and shown in Table 9 in last column. Precision is calculated as below

$$\text{Precision} = \frac{\text{Number of real defects}}{\text{Number of warnings}}$$

In table below, title of column 1 and 2 are as “A = Rule ID” and “B = System ID” respectively.

A	B	Total number of rule application	# of warnings generated	# of false positives	# of real defects found	DDR	Precision
1.	1	15	9	2	7	46.67 %	77.78%
	2	8	8	2	6	75.00 %	75.00%
	3	23	23	23	0	0.00%	0.00%
	4	10	10	6	4	40.00 %	40.00%
	5	16	16	10	6	37.50 %	37.50%
2.	1	15	9	2	7	46.67 %	77.78%
	2	8	2	0	2	25.00 %	100.00%
	3	23	23	23	0	0.00%	0.00%
	4	10	10	6	4	40.00 %	40.00%
	5	16	9	4	5	31.25 %	55.56%
3.	1	2	1	0	1	50.00 %	100.00%
	2	0	0	0	0	0.00%	0.00%
	3	0	0	0	0	0.00%	0.00%
	4	0	0	0	0	0.00%	0.00%
	5	0	0	0	0	0.00%	0.00%
4.	1	3	2	0	2	66.67 %	100.00%
	2	0	0	0	0	0.00%	0.00%
	3	6	0	0	0	0.00%	0.00%
	4	9	1	0	1	11.11 %	100.00%
	5	19	0	0	0	0.00%	0.00%
5.	1	5	0	0	0	0.00%	0.00%
	2	2	0	0	0	0.00%	0.00%
	3	23	2	0	2	8.70%	100.00%
	4	10	8	0	8	80.00 %	100.00%
	5	16	2	0	2	12.50 %	100.00%
6.	1	4	2	0	2	50.00	100.00%

A	B	Total number of rule application	# of warnings generated	# of false positives	# of real defects found	DDR	Precision
						%	
	2	2	1	0	1	50.00 %	100.00%
	3	23	0	0	0	0.00%	0.00%
	4	9	9	0	9	100.00 %	100.00%
	5	19	5	0	5	26.32 %	100.00%
7.	1	8	5	0	5	62.50 %	100.00%
	2	6	6	0	6	100.00 %	100.00%
	3	8	8	0	8	100.00 %	100.00%
	4	9	9	0	9	100.00 %	100.00%
	5	20	20	0	20	100.00 %	100.00%
8.	1	8	5	0	5	62.50 %	100.00%
	2	6	4	0	4	66.67 %	100.00%
	3	8	2	2	0	0.00%	0.00%
	4	9	4	0	4	44.44 %	100.00%
	5	20	6	1	5	25.00 %	83.33%
9.	1	8	5	0	5	62.50 %	100.00%
	2	6	4	0	4	66.67 %	100.00%
	3	8	2	2	0	0.00%	0.00%
	4	9	3	0	3	33.33 %	100.00%
	5	20	5	1	4	20.00 %	80.00%
10.	1	8	7	0	7	87.50 %	100.00%
	2	6	5	0	5	83.33 %	100.00%

A	B	Total number of rule application	# of warnings generated	# of false positives	# of real defects found	DDR	Precision
	3	8	2	2	0	0.00%	0.00%
	4	9	4	0	4	44.44 %	100.00%
	5	20	2	1	1	5.00%	50.00%
11.	1	8	4	0	4	50.00 %	100.00%
	2	6	2	0	2	33.33 %	100.00%
	3	8	1	1	0	0.00%	0.00%
	4	9	7	0	7	77.78 %	100.00%
	5	20	11	1	10	50.00 %	90.91%
Totals		**551**					

Table 9 Defects found against each rule

Following Table 12 shows total number of defects found with each rule per system and the total of all the defects found in the complete set of requirements used during the experiment.

Rule ID	System ID	Defects per system	Total defects found
1	1	7	24
	2	7	
	3	0	
	4	6	
	5	4	
2	1	7	18
	2	2	
	3	0	
	4	4	
	5	5	
3	1	1	1
	2	0	
	3	0	
	4	0	
	5	0	

4	1	2	3
	2	0	
	3	0	
	4	0	
	5	1	
5	1	0	12
	2	0	
	3	2	
	4	8	
	5	2	
6	1	2	17
	2	1	
	3	0	
	4	9	
	5	5	
7	1	5	48
	2	6	
	3	8	
	4	9	
	5	20	
8	1	5	18
	2	4	
	3	0	
	4	4	
	5	5	
9	1	5	16
	2	4	
	3	0	
	4	3	
	5	4	
10	1	7	17
	2	5	
	3	0	
	4	4	
	5	1	
11	1	4	23
	2	2	
	3	0	
	4	7	
	5	10	
Total defects			**197**

Table 10 Total number of defects found with each rule

The Figure 10 shows a bar chart based on table 12. This chart shows the defects found in all the requirement statements used in the project and the number of defects founds in each system. Figure 10 also provides a simple comparison of all the rules and performance of each rule.

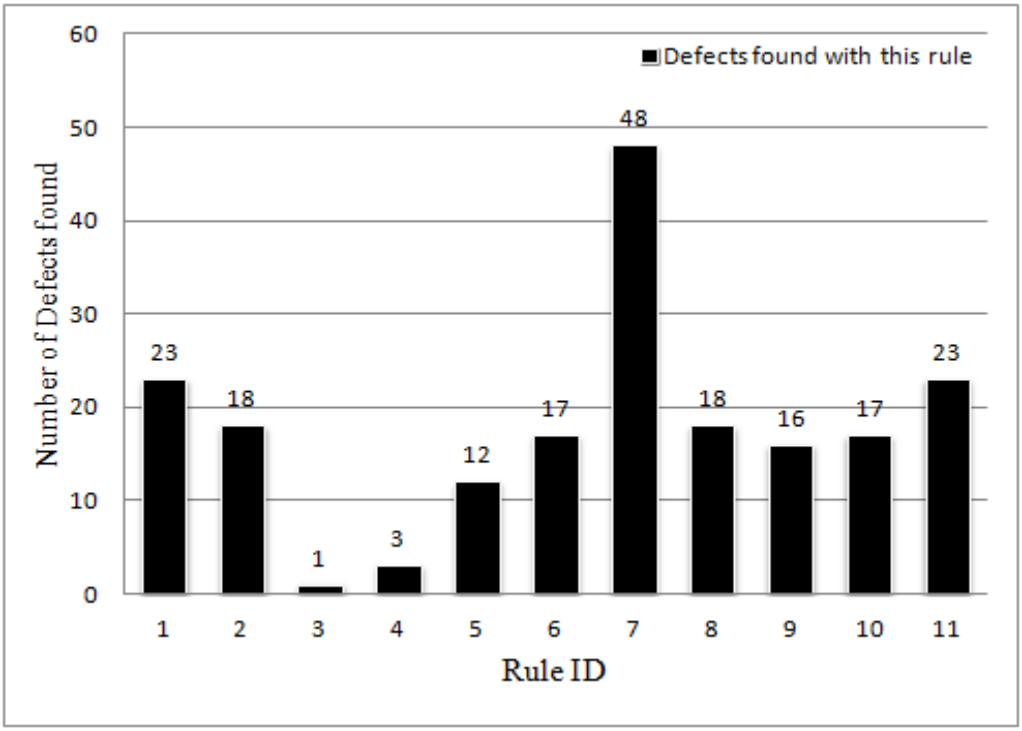

Figure 8 A bar chart showing defects found

The above figure also shows that the rule 7 has found the most number of defects in the database, whereas rule 1 is the next most successful rule. As obvious from Figure 10, rule 3 only found one defect. This has more to do with the bad practices used during the requirements phase i.e., writing of requirements. Rule 3 checks that every data validation has a related data output requirement statement, but the requirement documents used in the project did not had many data validations and were missing related data outputs. If the requirements

document has more data validations and is missing data output then Rule 3 will detect the missing requirements. Similar is the case in rule 7. Rule 7 checks that every usecase has a requirement of type **data validation**, but most of the usecases had no **data validation** requirement.

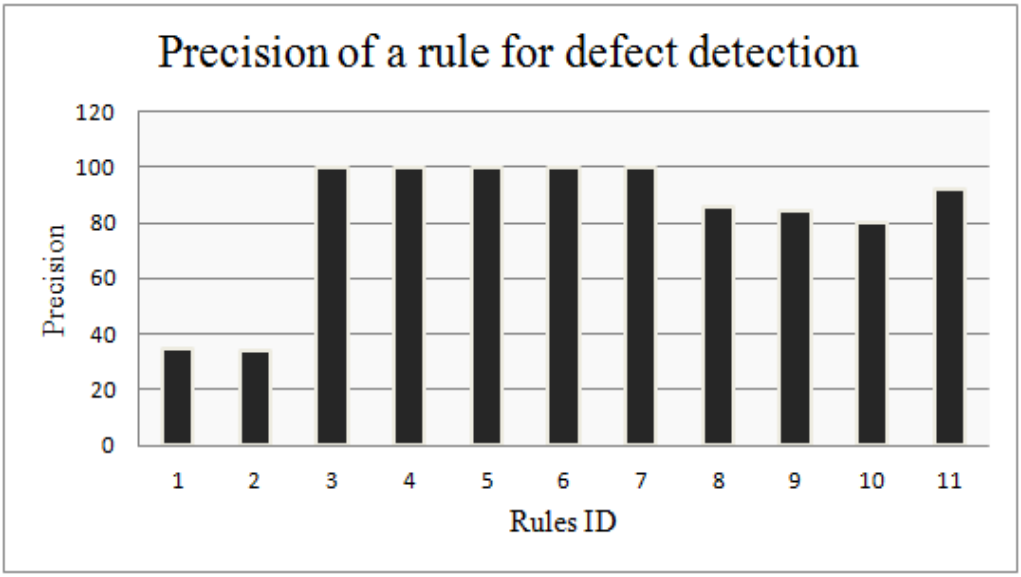

Figure 9 Precision of rules

Figure 9 shows a bar chart of precision of each rule. The precision is calculated according to precision formula given on page 43.

Table 11 shows ranking of rules, the ranking is based on defect detection rate of each rule and then on Precision of that rule. Table 11 shows that rule 7 has the highest defect detection rate and 100% precision. This makes rule 7 the highest ranking rule in the rules found.

Rank	Rule ID	# of rule application	# of warnings	# of false positives	Defects found	DDR	Precision
1.	7	51	48	0	48	94.11%	100.00%

Rank	Rule ID	# of rule application	# of warnings	# of false positives	Defects found	DDR	Precision
2.	3	2	1	0	1	50%	100.00%
3.	11	51	25	2	23	43.13%	92.00%
4.	8	51	21	3	18	35.29%	85.71%
5.	10	51	20	3	16	33.33%	80.00%
6.	1	72	66	43	23	31.94%	34.84%
7.	6	57	17	0	17	29.82%	100.00%
8.	9	51	19	3	16	29.41%	84.21%
9.	2	72	53	35	18	25.00%	33.96%
10.	5	56	12	0	12	21.42%	100.00%
11.	4	37	3	0	3	8.10%	100.00%
Total		**551**	**283**	**89**	**194**	**---**	**---**

Table 11 Ranking of rules

The above statistical data including Table 9, 10, 11, and Figure 10 can be used to create a report and can be used to develop a checklist, which then can be given to inspector. Inspectors can looks specifically into the found defects. After the inspection has been done then the final report of defects can be given back to authors of requirements, so that they can remove and repair defects and also validate the defects with customers.

After performing a complete validation on the database of 309 software requirement statements, I different set of 500 requirement statements from a new system was also validated. This was done to find out if the discovered rules are

complete? This is confirmed by finding out if the currently found requirement types are complete for enterprise domain. From the validation of new system, it is found that only one new requirement type is found. The new found requirement type is “post condition”; “A statement which describes the post condition of a usecase”. This can be a task for future efforts, i.e. to replicate experiments on requirement statements of enterprise software systems and thus improve requirement types set and rules.

6 FORMALIZATION

In this chapter all the software requirement inspection rules are represented in a formal logic.

6.1. Introduction

Natural languages (like, English) are ambiguous. A word, clause, or sentence can have multiple meanings in a natural language. In order to have the rule free of ambiguity, rules are converted to a formal language; Sentential Logic (SL) and Quantifier Logic (QL). SL is also known as propositional logic and QL is also known as predicate logic. Thus the rules shown in Table 13 are concise and free of ambiguity.

6.2. Formal logic representation

Table 12 gives the definition of symbols used in formal representation of software requirements inspection rules.

ID	Definitions
1.	R = set of atomic software requirements of a project
2.	DI = set of software requirements of type Data Input
3.	IDI = set of items in Data Input software requirements
4.	DV = set of software requirements of type Data Validation
5.	UC = set of Usecases/features
6.	$items$ = set of data items of a software requirements
7.	$Validates\ (x, y) = x$ validates y
8.	$Persistence\ (x, y) = x$ describes data persistence requirement of y

9.	$\boldsymbol{Output}\ (\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}$ **gives the output of** $\boldsymbol{y}$
10.	$\boldsymbol{Triggers}\ (\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}$ **triggers** $\boldsymbol{y}$
11.	$\boldsymbol{Complete}(\boldsymbol{x}) = \boldsymbol{x}$ **is complete**

Table 12 Requirement type representation

Table 13 shows the software requirement inspection rules from Table 7 represented into a formal language. Table 13 uses the formal language from predicate logic.

There is an important assumption about the 11th definition in Table 12. The function "Complete(x)" means the x is a complete requirement statement. Completeness cannot be validated algorithmically without human inspection.

Rule ID	Formal representation
1	$\forall x\ [x \in R\ \&\ x\ \in DI] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in DV]\ \&\ Validates\ (DV, DI\ )$
2	$\forall x\ [x \in R\ \&\ x\ \in DI] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in DP]\ \&\ Persistence\ (DP, DI\ )$
3	$\forall x\ [x \in R\ \&\ x\ \in DV] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in DO]\ \&\ Output\ (DO, DV\ )$
4	$\forall x\ [x \in R\ \&\ x\ \in ET] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in R]\ \&\ Triggers\ (ET, R\ )$
5	$\forall x\ [x \in R\ \&\ x\ \in DI] \rightarrow\ Complete\ (x)$
6	$\forall x\ [x \in R\ \&\ x\ \in DO] \rightarrow\ Complete\ (x)$
7	$\forall x\ [x\ \in UC] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in DV]$
8	$\forall x\ [x\ \in UC] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in DI]$
9	$\forall x\ [x\ \in UC] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in DO]$
10	$\forall x\ [x\ \in UC] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in BL]$
11	$\forall x\ [x\ \in UC] \rightarrow\ \exists y\ [y\ \in R\ \&\ y\ \in DP]$

Table 13 Formal representation of rules

7 CONCLUSION AND FUTURE WORK

Software requirements inspection can be beneficial in different stages of Software Development Life Cycle (SDLC). This thesis provided a review of three commonly used techniques for software requirements inspection and suggests an algorithmic technique with rules.

7.1 Summary

Chapter 1 presented an introduction of this thesis. It also discussed the importance of software requirements phase in SDLC.

Chapter 2 provided the background and literature review for software requirements inspection techniques. Three commonly used techniques are: Ad-hoc techniques, Checklist based technique, and scenario-based techniques. These techniques were discussed and empirical experiments performed by different researchers for comparing the three techniques were also presented.

The problem of efficiently inspecting software requirements is presented in Chapter 3. Chapter 4 described the proposed solution of the problem discussed in Chapter 3. An algorithmic technique for software requirements inspection is presented in Chapter 4. This chapter described the complete algorithm developed during the research project. This algorithm can provide better result in efficiently discovering defects in the software requirements.

Chapter 5 presented the validation of the algorithm. In the chapter, the details and results of the experiment performed are given; the algorithm was applied to a

set of software requirements. Tables and charts are given in the chapter showing the output data of the experiment.

Chapter 6 presented the 11 rules from chapter 4 in a formal language. This presentation in a formal language removes any ambiguity from the rules.

7.2 Conclusion

In a summarized list, the contributions by this thesis are:

- Comparison of currently practiced software requirements inspection techniques.
- Proposal of a systematic software requirements inspection technique.
- The formalization of rules of proposed algorithm.

7.2.1. Problems encountered

There were no major impediments faced during the project.

One issue during the requirements collection phase was that some requirement statements were not atomic, whereas some were not completely defined. Non-atomic statements were more of concern as such statements became difficult to categorize during the classification step of our algorithm.

7.3 Future work & Recommendation

We found that a domain-specific systematic technique can be better and efficient for performing software requirements inspection. This thesis focused on the software requirements of commercial enterprise domain. Future works can be

focused towards other domains, e.g. web development, embedded, real-time, medical, mobile, etc.

The set of rules for software requirement inspection developed during this project can be improved and new rules can be added to the set.

REFERENCES

[1] Ian Sommerville, *Software engineering*, 8th ed. Essex, UK: Pearson Education Ltd., 2007.

[2] Roger S. Pressman, *Software engineering A practitioner's approach*, 7th ed. New york, USA: Mc Graw-Hill, 2010.

[3] Pamela Zave, "Classification of research efforts in requirements engineering," *ACM Computing Surveys (CSUR)*, vol. 29, no. 4, pp. 315-321, Dec 1997.

[4] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," in *ICSE '00 Proceedings of the Conference on The Future of Software Engineering*, New York, 2000, pp. 35-46.

[5] R. Stevens, P. Brook, K. Jackson, and S. Arnold, *Systems engineering: coping with complexity*.: Prentice Hall, 1998.

[6] Philip B Crosby, *Quality Without Tears: The Art of Hassle-Free Management*, 1st ed. New York, USA: McGraw-Hill, 1995.

[7] E. Hull, K. Jackson, and J. Dick, *Requirements engineering*, 2nd ed. london, UK: Springer Verlag, 2005.

[8] B. Boehm and V.R. Basili, "Top 10 list [software development]," *IEEE Computer*, vol. 34, no. 1, pp. 135-137, 2001.

[9] Betty H. C. Cheng and Joanne M. Atlee., "Research directions in requirements engineering," *Future of Software Engineering, 2007.FOSE'07*, pp. 285-303, May 2007.

[10] M. Sagheb-Tehrani and A. Ghazarian, "Software development process: strategies for handling business rules and requirements," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 2, pp. 58-62, March 2002.

[11] Ian Sommerville and Pete Sawyer, *Requirements Engineering: A good practice guide*. West Sussex, UK: John Wiley & sons Ltd., 1997.

[12] M.E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 15, no. 3, pp. 182-211, 1976.

[13] Eric J. Braude, *Software engineering: an Object-Oriented perspective*. New York, USA: John Wiley & sons, Inc., 2001.

[14] A.A. Porter and L.G. Votta, "An experiment to assess different defect detection methods for software requirements inspections," in *ICSE '94 Proceedings of the 16th international conference on Software engineering*, Los Alamitos, CA, USA, 1994, pp. 103-112.

[15] P. Runeson, C. Andersson, T. Thelin, A. Andrews, and T. Berling, "What do we know about defect detection methods?," *Software, IEEE*, vol. 23, no. 3, pp. 82-90, May-June 2006.

[16] Watts S. Humphrey, *A discipline for software engineering*. Reading, Mass., USA: Addison-Wesley, 1995.

[17] L.G. Votta Jr, "Does every inspection need a meeting?," *ACM SIGSOFT Software Engineering Notes*, vol. 18, no. 5, pp. 107-114, December 1993.

[18] E. Kamsties, D.M. Berry, and B. Paech, "Detecting ambiguities in requirements documents using inspections," in *Workshop on Inspection in Software Engineering (WISE'01)*, Paris, France, 2001, pp. 68-80.

[19] G. Kotonya and I. Sommerville, *Requirements engineering*. West sussex, UK: Wiley Chichester, 1998.

[20] B.W. Boehm, "Verifying and validating software requirements and design specifications," *IEEE Software*, vol. 1, no. 1, pp. 75-88, January 1984.

[21] David A. Cook. (2002, May) Software Technology Conference (STC 2002). [Online]. http://sstc-online.org/2002/SpkrPDFS/ThrTracs/p961.pdf

[22] A. Terry Bahill and Steven J. Henderson, "Requirements development, verification, and validation exhibited in famous failures," *Systems engineering*, vol. 8, no. 1, pp. 1-14, 2005.

[23] Shari Lawrence Pfleeger and Joanne M. Atlee, *Software Engineering: Theory and Practice*, 4th ed. New Jersey, USA: Pearson Higher Education, 2010.

[24] H.F. Hofmann and F. Lehner, "Requirements engineering as a success factor in software projects," *IEEE Software*, vol. 18, no. 4, pp. 58-66, July/August 2001.

[25] Ralph D. Jeffords and Constance L. Heitmeyer, "A Strategy for Efficiently Verifying Requirements Specifications Using Composition and Invariants," *11th ACM SIGSOFT international symposium on Foundations of Software Engineering*, vol. 28, no. 5, pp. 28-37, September 2003.

[26] C.L. Heitmeyer, "Software cost reduction," *Encyclopedia of Software Engineering*, January 2002.

[27] B.W. Boehm, "Software engineering economics," *Software Engineering, IEEE Transactions on*, vol. SE-10, no. 1, pp. 200-217, January 1984.

[28] B. Regnell, P. Runeson, and T. Thelin, "Are the perspectives really different?–Further experimentation on scenario-based reading of requirements," *Empirical Software Engineering*, vol. 5, no. 4, pp. 331-356, 2000.

[29] A.A. Porter, L.G. Votta Jr, and V.R. Basili, "Comparing detection methods for software requirements inspections: a replicated experiment," *Software Engineering, IEEE Transactions on* , vol. 21, no. 6, pp. 563-575, 1995.

[30] B. Cheng and R. Jeffery, "Comparing inspection strategies for software requirement specifications," in *Software Engineering Conference, 1996. Proceedings of 1996 Australian*, Melbourne, Vic. , Australia, 1996, pp. 203-211.

[31] P. Fusaro, F. Lanubile, and G. Visaggio, "A replicated experiment to assess requirements inspection techniques," *Empirical Software Engineering*, vol. 2, no. 1, pp. 39-57, 1997.

[32] Kristian Sandahl et al., "An Extended Replication of an Experiment for Assessing Methods for Software Requirements Inspections," *Empirical Software Engineering*, vol. 3, no. 4, pp. 327-354, December 1998.

[33] F. Lanubile and G. Visaggio, Evaluating defect detection techniques for software requirements inspections, 2000, citeseer.

[34] Victor Basili et al. Lab Package for the Empirical Investigation of Perspective-Based Reading. [Online]. http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/pbr_package/manual.html

[35] B. Brykczynski, "A survey of software inspection checklists," *ACM SIGSOFT Software Engineering Notes*, vol. 24, no. 1, p. 82, January 1999.

[36] G. Michael Schneider, Johnny Martin, and W. T. Tsai, "An experimental study of fault detection in user requirements documents," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 1, no. 2, pp. 189-204, April 1992.

[37] J. Martin and W.T. Tsai, "NFold Inspection: A Requirements Analysis Technique," *Communications of the ACM*, vol. 33, no. 2, pp. 225-232, February 1990.

[38] Robyn R. Lutz, "Targeting Safety-Related Errors During Software Requirements Analysis," in *Proceeding SIGSOFT '93 Proceedings of the 1st ACM SIGSOFT symposium on Foundations of software engineering*, Los Angeles, 1993, pp. 99-106.

[39] M. Halling, P. Grunbacher, and S. Biffl, "Tailoring a COTS Group Support System for Software Requirements Inspection ," in *Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on* , San Diego,CA, 2001, pp. 201-208.

[40] M. Jarke, X.T. Bui, and J.M. Carroll, "Scenario management: An interdisciplinary approach," *Requirements Engineering*, vol. 3, no. 3, pp. 155-173, 1998.

[41] V.R. Basili et al., "The empirical investigation of perspective-based reading," *Empirical Software Engineering*, vol. 1, no. 2, pp. 133-164, 1996.

[42] F. Shull, I. Rus, and V. Basili, "How perspective-based reading can improve requirements inspections," *IEEE Computer*, vol. 33, no. 7, pp. 73-79, August 2000.

[43] Annie I. Antón and Colin Potts, "The Use of Goals to Surface Requirements for Evolving Systems," in *Proceedings of the 20th international conference on Software engineering (ICSE '98)*, Kyoto, Japan, 1998, pp. 157-166.

[44] M. Mannio and Uolevi Nikula, "Requirements Elicitation Using a Combination of Prototypes and scenarios," in *Workshop on Requirements Engineering*, Buenos Aires, Argentina, 2001, pp. 283-297.

[45] David L. Parnas and David M. Weiss, "Active design reviews: principles and practices," in *Proceedings of the 8th international conference on Software engineering*, London, United Kingdom, 1985, pp. 132-136.

[46] A.A. Porter, L.G. Votta Jr, and V.R. Basili, "Comparing detection methods for software requirements inspections: A replicated experiment," *Software Engineering, IEEE Transactions on*, vol. 21, no. 6, pp. 563-575, June 1995.

[47] Barrett R. Bryant, Jeff Gray, and Marjan Mernik, "Domain-specific software engineering," in *FoSER '10 Proceedings of the FSE/SDP workshop on Future of software engineering research*, Santa Fe, New Mexico, USA, 2010, p. 65.

[48] A. Eberlein, and M. Moussavi M. Galster, "Atomic Requirements for Software Architecting," in *SEA '07 Proceedings of the 11th IASTED International Conference on Software Engineering and Applications*, Anaheim, CA, 2007, pp. 143-147.

[49] H Salzer, "ATRs (Atomic Requirements) Used Throughout Development Lifecycle," in *12th INTERNATIONAL SOFTWARE QUALITY WEEK (QW'99)*, San Jose, CA, USA , 1999.

APPENDIX A

REQUIREMENT TYPES

Table 14 gives the requirement types discovered in requirements database.

ID	Requirement type	Description
1	Data Input	Data entered into the system by the actor of the use case
2	Data Output	The intermediate or final result of the use case outputted by the system on the screen/printer. The content of the screens and the rules for displaying those contents.
3	Data Validation	validation of data items inputted by the actors of the use cases
4	Business Logic	Application or business logic including calculations
5	Data Persistence	All database related operations including reading, updating, inserting and deleting from/to a database
6	Messaging	Sending an email to a party or a message sent from one system/component of system to another. Also describes the content of Email.
7	Event Trigger	Actor clicks on a menu item or link or button - a command
8	User Interface Navigation	Flow of application's screen. (Transition between screens). The rules for the transitions between screens.
9	User Interface	The layout of the page and screen e.g. an input form

ID	Requirement type	Description
10	External Call	Calls/messages between different systems./Parameters used to make a call to system and values received from system.
11	High Level Requirement	A feature/capability/high level requirement that should be broken down into atomic requirements.
12	User Interface Logic	User interface/interaction behavior
13	External Behavior	Explains the behavior of an external 3rd party system.

Table 14 Requirement types

Table 16 shows the total number of each requirement type found in the database used in our experiment.

Req. ID	Number of requirements
1.	60
2.	61
3.	2
4.	40
5.	51
6.	0
7.	38
8.	24
9.	15
10.	4
11.	4
12.	8
13.	2
Total requirements	**309**

Table 15 Number of requirement of each type

APPENDIX B

DATA OF REQUIREMENT INSPECTION WITH RULES

All the requirements in the database were inspected against the rules, given in Table 9 in section 4.2.2.2 Following table shows each requirement statement with applied rule and the defect found or in some cases, the requirement qualified the rule and there was not defect.

The titles of each column in below table are defined as: Rule ID = Rule ID, Req. ID = Requirement ID, Sys. ID = System ID, related req. = related requirement, d? = defect found or not (1 shows that defect is found), Comment = comment if the defect found is a false positive and not an actual defect, name of last column "Defect type" is self explanatory.

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
1	1	431	1		1	False positive	
2	1	431	1		1	False positive	
3	1	434	1	436			
4	1	434	1	436			
5	1	434	1	436			
6	1	434	1	436			
7	1	434	1	436			
8	1	435	1		1		Missing requirement
9	1	439	1	442			
10	1	447	1		1		Missing requirement
11	1	447	1		1		Missing requirement
12	1	447	1		1		Missing

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
							requirement
13	1	447	1		1		Missing requirement
14	1	447	1		1		Missing requirement
15	1	447	1		1		Missing requirement
16	1	461	2		1	False positive	
17	1	461	2		1	False positive	
18	1	473	2		1		Missing requirement
19	1	473	2		1		Missing requirement
20	1	473	2		1		Missing requirement
21	1	473	2		1		Missing requirement
22	1	473	2		1		Missing requirement
23	1	473	2		1		Missing requirement
24	1	589	3		1	False positive	
25	1	592	3		1	False positive	
26	1	597	3		1	False positive	
27	1	604	3		1	False positive	
28	1	607	3		1	False positive	
29	1	611	3		1	False positive	
30	1	615	3		1	False positive	

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
31	1	618	3		1	False positive	
32	1	621	3		1	False positive	
33	1	628	3		1	False positive	
34	1	631	3		1	False positive	
35	1	634	3		1	False positive	
36	1	637	3		1	False positive	
37	1	641	3		1	False positive	
38	1	644	3		1	False positive	
39	1	647	3		1	False positive	
40	1	651	3		1	False positive	
41	1	654	3		1	False positive	
42	1	658	3		1	False positive	
43	1	664	3		1	False positive	
44	1	667	3		1	False positive	
45	1	670	3		1	False positive	
46	1	672	3		1	False positive	
47	1	1016	4		1	False positive	
48	1	1021	4		1		Missing requirement
49	1	1025	4		1	False positive	
50	1	1032	4		1	False positive	
51	1	1037	4		1		Missing requirement
52	1	1039	4		1	False positive	
53	1	1043	4		1		Missing requirement
54	1	1046	4		1	False positive	
55	1	1052	4		1		Missing

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
							requirement
56	1	1056	4		1	False positive	
57	1	1092	5		1		Missing requirement
58	1	1098	5		1	False positive	
59	1	1103	5		1		Missing requirement
60	1	1112	5		1	False positive	
61	1	1120	5		1	False positive	
62	1	1129	5		1	False positive	
63	1	1141	5		1	False positive	
64	1	1150	5		1		Missing requirement
65	1	1156	5		1	False positive	
66	1	1162	5		1	False positive	
67	1	1170	5		1		Missing requirement
68	1	1171	5		1		Missing requirement
69	1	1178	5		1	False positive	
70	1	1199	5		1	False positive	
71	1	1204	5		1		Missing requirement
72	1	1210	5		1	False positive	
73	2	431	1		1		Missing requirement
74	2	431	1		1		Missing requirement

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
75	2	434	1		1		Missing requirement
76	2	434	1		1		Missing requirement
77	2	434	1		1		Missing requirement
78	2	434	1		1		Missing requirement
79	2	434	1		1		Missing requirement
80	2	435	1		1	False positive	
81	2	439	1		1	False positive	
82	2	447	1	448			
83	2	447	1	448			
84	2	447	1	448			
85	2	447	1	448			
86	2	447	1	448			
87	2	447	1	448			
88	2	461	2		1		Missing requirement
89	2	461	2		1		Missing requirement
90	2	473	2	474			
91	2	473	2	474			
92	2	473	2	474			
93	2	473	2	474			
94	2	473	2	474			
95	2	473	2	474			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
96	2	589	3		1	False positive	
97	2	592	3		1	False positive	
98	2	597	3		1	False positive	
99	2	604	3		1	False positive	
100	2	607	3		1	False positive	
101	2	611	3		1	False positive	
102	2	615	3		1	False positive	
103	2	618	3		1	False positive	
104	2	621	3		1	False positive	
105	2	628	3		1	False positive	
106	2	631	3		1	False positive	
107	2	634	3		1	False positive	
108	2	637	3		1	False positive	
109	2	641	3		1	False positive	
110	2	644	3		1	False positive	
111	2	647	3		1	False positive	

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
112	2	651	3		1	False positive	
113	2	654	3		1	False positive	
114	2	658	3		1	False positive	
115	2	664	3		1	False positive	
116	2	667	3		1	False positive	
117	2	670	3		1	False positive	
118	2	672	3		1	False positive	
119	2	1016	4		1	False positive	
120	2	1021	4		1		Missing requirement
121	2	1025	4		1	False positive	
122	2	1032	4		1	False positive	
123	2	1037	4		1		Missing requirement
124	2	1039	4		1	False positive	
125	2	1043	4		1		Missing requirement

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
126	2	1046	4		1	False positive	
127	2	1052	4		1		Missing requirement
128	2	1056	4		1	False positive	
129	2	1092	5		1		Missing requirement
130	2	1098	5		1		Missing requirement
131	2	1103	5	1108			
132	2	1112	5	1115			
133	2	1120	5	1123			
134	2	1129	5	1132			
135	2	1141	5		1		Missing requirement
136	2	1150	5		1	False positive	
137	2	1156	5	1157			
138	2	1162	5	1164			
139	2	1170	5		1		Missing requirement

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
140	2	1171	5		1		Missing requirement
141	2	1178	5	1181			
142	2	1199	5		1	False positive	
143	2	1204	5		1	False positive	
144	2	1210	5		1	False positive	
145	3	436	1		1		Missing requirement
146	3	442	1	442			
147	4	444	1	445			
148	4	459	1		1		Missing requirement
149	4	460	1		1		Missing requirement
150	4	598	3	599			
151	4	612	3	613			
152	4	625	3	626			
153	4	638	3	639			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
154	4	655	3	656			
155	4	673	3	674			
156	4	1018	4	1019			
157	4	1030	4	1031			
158	4	1034	4	1035			
159	4	1036	4		1		Missing requirement
160	4	1041	4	1042			
161	4	1044	4	1045			
162	4	1048	4	1049			
163	4	1050	4	1051			
164	4	1060	4	1061			
165	4	1095	5	1096			
166	4	1101	5	1102			
167	4	1104	5	1105			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
168	4	1113	5	1114			
169	4	1121	5	1122			
170	4	1130	5	1131			
171	4	1137	5	1138			
172	4	1144	5	1145			
173	4	1148	5	1149			
174	4	1151	5	1152			
175	4	1169	5	1170			
176	4	1176	5	1177			
177	4	1179	5	1180			
178	4	1186	5	1187			
179	4	1190	5	1191			
180	4	1196	5	1197			
181	4	1202	5	1203			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
182	4	1205	5	1206			
183	4	1209	5	1210			
184	5	431	1				
185	5	434	1				
186	5	435	1				
187	5	439	1				
188	5	447	1				
189	5	461	2				
190	5	473	2				
191	5	589	3		1	underspecified	Incomplete Req.
192	5	592	3				
193	5	597	3				
194	5	604	3				
195	5	607	3				

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
196	5	611	3				
197	5	615	3		1	underspecified	Incomplete Req.
198	5	618	3				
199	5	621	3				
200	5	628	3				
201	5	631	3				
202	5	634	3				
203	5	637	3				
204	5	641	3				
205	5	644	3				
206	5	647	3				
207	5	651	3				
208	5	654	3				
209	5	658	3				

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
210	5	664	3				
211	5	667	3				
212	5	670	3				
213	5	672	3				
214	5	1016	4		1	underspecified	Incomplete Req.
215	5	1021	4		1	underspecified	Incomplete Req.
216	5	1025	4		1	underspecified	Incomplete Req.
217	5	1032	4		1	underspecified	Incomplete Req.
218	5	1037	4		1	underspecified	Incomplete Req.
219	5	1039	4		1	underspecified	Incomplete Req.
220	5	1043	4				
221	5	1046	4		1	underspecified	Incomplete Req.
222	5	1052	4				
223	5	1056	4		1	underspecified	Incomplete Req.

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
224	5	1092	5		1	underspecified	Incomplete Req.
225	5	1098	5		1	underspecified	Incomplete Req.
226	5	1103	5				
227	5	1112	5				
228	5	1120	5				
229	5	1129	5				
230	5	1141	5				
231	5	1150	5				
232	5	1156	5				
233	5	1162	5				
234	5	1170	5				
235	5	1171	5				
236	5	1178	5				
237	5	1199	5				

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
238	5	1204	5				
239	5	1210	5				
240	6	432	1		1	underspecified	Incomplete
241	6	441	1		1	underspecified	Incomplete
242	6	443	1				
243	6	450	1				
244	6	462	2		1	underspecified	Incomplete
245	6	476	2				
246	6	591	3				
247	6	594	3				
248	6	603	3				
249	6	606	3				
250	6	610	3				
251	6	617	3				

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
252	6	620	3				
253	6	623	3				
254	6	626	3				
255	6	630	3				
256	6	633	3				
257	6	636	3				
258	6	639	3				
259	6	643	3				
260	6	646	3				
261	6	650	3				
262	6	653	3				
263	6	656	3				
264	6	660	3				
265	6	663	3				

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
266	6	666	3				
267	6	669	3				
268	6	674	3				
269	6	1017	4		1	underspecified	Incomplete
270	6	1019	4		1	underspecified	Incomplete
271	6	1023	4		1	underspecified	Incomplete
272	6	1031	4		1	underspecified	Incomplete
273	6	1035	4		1	underspecified	Incomplete
274	6	1040	4		1	underspecified	Incomplete
275	6	1049	4		1	underspecified	Incomplete
276	6	1057	4		1	underspecified	Incomplete
277	6	1061	4		1	underspecified	Incomplete
278	6	1093	5		1	underspecified	Incomplete
279	6	1097	5				

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
280	6	1106	5		1	underspecified	Incomplete
281	6	1111	5				
282	6	1114	5				
283	6	1119	5				
284	6	1125	5				
285	6	1134	5				
286	6	1138	5				
287	6	1142	5				
288	6	1155	5				
289	6	1158	5				
290	6	1159	5				
291	6	1165	5				
292	6	1177	5				
293	6	1192	5				

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
294	6	1193	5		1	underspecified	Incomplete
295	6	1197	5		1	underspecified	Incomplete
296	6	1206	5		1	underspecified	Incomplete
297	7	39(Feature ID)	1		1		Missing requirement
298	7	40(Feature ID)	1	444			
299	7	41(Feature ID)	1		1		Missing requirement
300	7	42(Feature ID)	1		1		Missing requirement
301	7	43(Feature ID)	1		1		Missing requirement
302	7	44(Feature ID)	1		1		Missing requirement
303	7	45(Feature ID)	1	459			
304	7	46(Feature ID)	1	460			
305	7	47(Feature ID)	2		1		Missing requirement
306	7	48(Feature ID)	2		1		Missing requirement
307	7	49(Feature ID)	2		1		Missing requirement

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
308	7	50(Feature ID)	2		1		Missing requirement
309	7	51(Feature ID)	2		1		Missing requirement
310	7	52(Feature ID)	2		1		Missing requirement
311	7	72(Feature ID)	3		1		Missing requirement
312	7	73(Feature ID)	3		1		Missing requirement
313	7	74(Feature ID)	3		1		Missing requirement
314	7	75(Feature ID)	3		1		Missing requirement
315	7	76(Feature ID)	3		1		Missing requirement
316	7	77(Feature ID)	3		1		Missing requirement
317	7	78(Feature ID)	3		1		Missing requirement
318	7	79(Feature ID)	3		1		Missing requirement
319	7	129(Feature ID)	4		1		Missing requirement
320	7	130(Feature ID)	4		1		Missing requirement
321	7	131(Feature ID)	4		1		Missing requirement

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
322	7	132(Feature ID)	4		1		Missing requirement
323	7	133(Feature ID)	4		1		Missing requirement
324	7	134(Feature ID)	4		1		Missing requirement
325	7	135(Feature ID)	4		1		Missing requirement
326	7	136(Feature ID)	4		1		Missing requirement
327	7	137(Feature ID)	4		1		Missing requirement
328	7	145(Feature ID)	5		1		Missing requirement
329	7	146(Feature ID)	5		1		Missing requirement
330	7	147(Feature ID)	5		1		Missing requirement
331	7	148(Feature ID)	5		1		Missing requirement
332	7	149(Feature ID)	5		1		Missing requirement
333	7	150(Feature ID)	5		1		Missing requirement
334	7	151(Feature ID)	5		1		Missing requirement
335	7	152(Feature ID)	5		1		Missing requirement

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
336	7	153(Feature ID)	5		1		Missing requirement
337	7	154(Feature ID)	5		1		Missing requirement
338	7	155(Feature ID)	5		1		Missing requirement
339	7	156(Feature ID)	5		1		Missing requirement
340	7	157(Feature ID)	5		1		Missing requirement
341	7	158(Feature ID)	5		1		Missing requirement
342	7	159(Feature ID)	5		1		Missing requirement
343	7	160(Feature ID)	5		1		Missing requirement
344	7	161(Feature ID)	5		1		Missing requirement
345	7	162(Feature ID)	5		1		Missing requirement
346	7	163(Feature ID)	5		1		Missing requirement
347	7	164(Feature ID)	5		1		Missing requirement
348	8	39(Feature ID)	1	431			
349	8	40(Feature ID)	1	434			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
350	8	41(Feature ID)	1	447			
351	8	42(Feature ID)	1		1		Missing requirement
352	8	43(Feature ID)	1		1		Missing requirement
353	8	44(Feature ID)	1		1		Missing requirement
354	8	45(Feature ID)	1		1		Missing requirement
355	8	46(Feature ID)	1		1		Missing requirement
356	8	47(Feature ID)	2	461			
357	8	48(Feature ID)	2		1		Missing requirement
358	8	49(Feature ID)	2	473			
359	8	50(Feature ID)	2		1		Missing requirement
360	8	51(Feature ID)	2		1		Missing requirement
361	8	52(Feature ID)	2		1		Missing requirement
362	8	72(Feature ID)	3	589			
363	8	73(Feature ID)	3	601			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
364	8	74(Feature ID)	3	615			
365	8	75(Feature ID)	3			False positive	this feature is missing in DB
366	8	76(Feature ID)	3	628			
367	8	77(Feature ID)	3	641			
368	8	78(Feature ID)	3	658			
369	8	79(Feature ID)	3			False positive	this feature is missing in DB
370	8	129(Feature ID)	4		1		Missing requirement
371	8	130(Feature ID)	4	1016			
372	8	131(Feature ID)	4		1		Missing requirement
373	8	132(Feature ID)	4		1		Missing requirement
374	8	133(Feature ID)	4	1025			
375	8	134(Feature ID)	4	1032			
376	8	135(Feature ID)	4	1039			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
377	8	136(Feature ID)	4		1		Missing requirement
378	8	137(Feature ID)	4	1056			
379	8	145(Feature ID)	5	1092			
380	8	146(Feature ID)	5	1100			
381	8	147(Feature ID)	5	1112			
382	8	148(Feature ID)	5	1120			
383	8	149(Feature ID)	5	1129			
384	8	150(Feature ID)	5		1		Missing requirement
385	8	151(Feature ID)	5	1141			
386	8	152(Feature ID)	5		1		Missing requirement
387	8	153(Feature ID)	5	1150			
388	8	154(Feature ID)	5	1156			
389	8	155(Feature ID)	5	1162			
390	8	156(Feature ID)	5	1170			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
391	8	157(Feature ID)	5	1178			
392	8	158(Feature ID)	5			False positive	this feature is missing in DB
393	8	159(Feature ID)	5		1		Missing requirement
394	8	160(Feature ID)	5		1		Missing requirement
395	8	161(Feature ID)	5		1		Missing requirement
396	8	162(Feature ID)	5	1199			
397	8	163(Feature ID)	5	1204			
398	8	164(Feature ID)	5	1210			
399	9	39(Feature ID)	1	432			
400	9	40(Feature ID)	1	443			
401	9	41(Feature ID)	1	450			
402	9	42(Feature ID)	1		1		Missing requirement
403	9	43(Feature ID)	1		1		Missing requirement
40	9	44(Feature	1		1		Missing

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
4		ID)					requirement
405	9	45(Feature ID)	1		1		Missing requirement
406	9	46(Feature ID)	1		1		Missing requirement
407	9	47(Feature ID)	2	462			
408	9	48(Feature ID)	2		1		Missing requirement
409	9	49(Feature ID)	2	476			
410	9	50(Feature ID)	2		1		Missing requirement
411	9	51(Feature ID)	2		1		Missing requirement
412	9	52(Feature ID)	2		1		Missing requirement
413	9	72(Feature ID)	3	591			
414	9	73(Feature ID)	3	603			
415	9	74(Feature ID)	3	617			
416	9	75(Feature ID)	3			False positive	this feature is missing in DB
417	9	76(Feature ID)	3	630			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
418	9	77(Feature ID)	3	643			
419	9	78(Feature ID)	3	660			
420	9	79(Feature ID)	3			False positive	this feature is missing in DB
421	9	129(Feature ID)	4		1		Missing requirement
422	9	130(Feature ID)	4	1017			
423	9	131(Feature ID)	4		1		Missing requirement
424	9	132(Feature ID)	4	1023			
425	9	133(Feature ID)	4	1026			
426	9	134(Feature ID)	4	1033			
427	9	135(Feature ID)	4	1040			
428	9	136(Feature ID)	4		1		Missing requirement
429	9	137(Feature ID)	4	1057			
430	9	145(Feature ID)	5	1093			
43	9	146(Featur	5	1106			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
1		e ID)					
432	9	147(Feature ID)	5	1111			
433	9	148(Feature ID)	5	1119			
434	9	149(Feature ID)	5	1134			
435	9	150(Feature ID)	5	1138			
436	9	151(Feature ID)	5	1142			
437	9	152(Feature ID)	5		1		Missing requirement
438	9	153(Feature ID)	5		1		Missing requirement
439	9	154(Feature ID)	5	1159			
440	9	155(Feature ID)	5	1165			
441	9	156(Feature ID)	5		1		Missing requirement
442	9	157(Feature ID)	5	1117			
443	9	158(Feature ID)	5			False positive	this feature is missing in DB
444	9	159(Feature ID)	5	1192			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
445	9	160(Feature ID)	5	1193			
446	9	161(Feature ID)	5	1197			
447	9	162(Feature ID)	5		1		
448	9	163(Feature ID)	5	1206			
449	9	164(Feature ID)	5		1		Missing requirement
450	10	39(Feature ID)	1		1		Missing requirement
451	10	40(Feature ID)	1	440			
452	10	41(Feature ID)	1		1		Missing requirement
453	10	42(Feature ID)	1		1		Missing requirement
454	10	43(Feature ID)	1		1		Missing requirement
455	10	44(Feature ID)	1		1		Missing requirement
456	10	45(Feature ID)	1		1		Missing requirement
457	10	46(Feature ID)	1		1		Missing requirement
458	10	47(Feature ID)	2		1		Missing requirement

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
459	10	48(Feature ID)	2	470			
460	10	49(Feature ID)	2		1		Missing requirement
461	10	50(Feature ID)	2		1		Missing requirement
462	10	51(Feature ID)	2		1		Missing requirement
463	10	52(Feature ID)	2		1		Missing requirement
464	10	72(Feature ID)	3	588			
465	10	73(Feature ID)	3	600			
466	10	74(Feature ID)	3	614			
467	10	75(Feature ID)	3			False positive	this feature is missing in DB
468	10	76(Feature ID)	3	627			
469	10	77(Feature ID)	3	640			
470	10	78(Feature ID)	3	657			
471	10	79(Feature ID)	3			False positive	this feature is missing in DB

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
472	10	129(Feature ID)	4	1014			
473	10	130(Feature ID)	4	1015			
474	10	131(Feature ID)	4		1		Missing requirement
475	10	132(Feature ID)	4	1024			
476	10	133(Feature ID)	4		1		Missing requirement
477	10	134(Feature ID)	4		1		Missing requirement
478	10	135(Feature ID)	4	1038			
479	10	136(Feature ID)	4	1055			
480	10	137(Feature ID)	4		1		Missing requirement
481	10	145(Feature ID)	5		1		Missing requirement
482	10	146(Feature ID)	5	1099			
483	10	147(Feature ID)	5	1109			
484	10	148(Feature ID)	5	1117			
485	10	149(Feature ID)	5	1126			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
486	10	150(Feature ID)	5	1136			
487	10	151(Feature ID)	5	1139			
488	10	152(Feature ID)	5	1143			
489	10	153(Feature ID)	5	1146			
490	10	154(Feature ID)	5	1153			
491	10	155(Feature ID)	5	1162			
492	10	156(Feature ID)	5	1167			
493	10	157(Feature ID)	5	1173			
494	10	158(Feature ID)	5			False positive	this feature is missing in DB
495	10	159(Feature ID)	5	1184			
496	10	160(Feature ID)	5	1188			
497	10	161(Feature ID)	5	1194			
498	10	162(Feature ID)	5	1198			
49	10	163(Featur	5	1201			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
9		e ID)					
500	10	164(Feature ID)	5	1207			
501	11	39(Feature ID)	1		1		Missing requirement
502	11	40(Feature ID)	1		1		Missing requirement
503	11	41(Feature ID)	1	448			
504	11	42(Feature ID)	1	454			
505	11	43(Feature ID)	1	455			
506	11	44(Feature ID)	1	457			
507	11	45(Feature ID)	1		1		Missing requirement
508	11	46(Feature ID)	1		1		Missing requirement
509	11	47(Feature ID)	2		1		Missing requirement
510	11	48(Feature ID)	2	464			
511	11	49(Feature ID)	2	474			
512	11	50(Feature ID)	2		1		Missing requirement
51	11	51(Feature	2	482			

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
3		ID)					
514	11	52(Feature ID)	2	483			
515	11	72(Feature ID)	3	590			
516	11	73(Feature ID)	3	602			
517	11	74(Feature ID)	3	616			
518	11	75(Feature ID)	3			False positive	this feature is missing in DB
519	11	76(Feature ID)	3	629			
520	11	77(Feature ID)	3	642			
521	11	78(Feature ID)	3	659			
522	11	79(Feature ID)	3			False positive	this feature is missing in DB
523	11	129(Feature ID)	4		1		Missing requirement
524	11	130(Feature ID)	4		1		Missing requirement
525	11	131(Feature ID)	4		1		Missing requirement
52	11	132(Featur	4		1		Missing

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
6		e ID)					requirement
527	11	133(Feature ID)	4	1028			
528	11	134(Feature ID)	4		1		Missing requirement
529	11	135(Feature ID)	4	1045			
530	11	136(Feature ID)	4		1		Missing requirement
531	11	137(Feature ID)	4		1		Missing requirement
532	11	145(Feature ID)	5		1		Missing requirement
533	11	146(Feature ID)	5	1108			
534	11	147(Feature ID)	5	1115			
535	11	148(Feature ID)	5	1123			
536	11	149(Feature ID)	5	1132			
537	11	150(Feature ID)	5		1		Missing requirement
538	11	151(Feature ID)	5		1		Missing requirement
539	11	152(Feature ID)	5		1		Missing requirement
54	11	153(Featur	5		1		Missing

ID	Rule ID	Req. ID	Sys. ID	related Req.	d?	Comment	Defect type
0		e ID)					requirement
541	11	154(Feature ID)	5	1157			
542	11	155(Feature ID)	5	1164			
543	11	156(Feature ID)	5		1		Missing requirement
544	11	157(Feature ID)	5	1181			
545	11	158(Feature ID)	5			False positive	this feature is missing in DB
546	11	159(Feature ID)	5	1187			
547	11	160(Feature ID)	5	1191			
548	11	161(Feature ID)	5		1		Missing requirement
549	11	162(Feature ID)	5		1		Missing requirement
550	11	163(Feature ID)	5		1		Missing requirement
551	11	164(Feature ID)	5		1		Missing requirement

Table 16 All requirements inspected and rule applied with defects found