Requirements quality assessment for outsourcing

Sharma, A.

Award date:
2009

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science

Requirements Quality Assessment for Outsourcing

Arpit Sharma
(0666074)
Master Thesis

Supervisors:
dr. Alexander Serebrenik
ir. Martijn Klabbers

Eindhoven, The Netherlands
August 2009

Abstract

From the previous experience of various university projects and software development companies, it has been observed that poor quality of software requirements may result in failure of the project. This problem becomes more significant when the software projects are outsourced in order to reduce time to market and achieve cost cutting. Many outsourced projects have failed in the past due to the lack of high quality requirements or change in requirements that could not be reflected in software product.

Certification of software requirements or software artifacts in general, offers organizations more certainty and confidence about software. If an organization wants certainty about or confidence in a software artifact it can request a LaQuSo certificate. In order to certify the software artifacts created during different stages of software development, LaQuSo has designed a product certification model called LSPCM. This model can be used to analyze software requirements, detect the defects and when they are solved, certify these requirements.

The model has three major weaknesses. First is that it does not stress the issues involved in outsourcing that influence the quality of software requirements. Due to this fact it is not suitable for certifying software requirements of outsourcing projects. Secondly, the quality of the requirements cannot be measured objectively using LSPCM; for certification, the requirements must be analyzed by an experienced analyst. This means that LSPCM provides room for subjective interpretation. Thirdly, most of the conformance quality checks are performed manually in LSPCM, thus the amount of time required for quality assessment and certification is high.

Along with these major weaknesses the model also has three minor weaknesses. First it does not give weight to availability of project-related documents in calculating the overall certification level. Secondly, it is not sensitive to IT domains. Thirdly, certification level is given based on the type of verification performed (e.g., manual, automatic) on the requirements document which is not very sensible in actual practice.

These weaknesses need to be tackled in order to make LSPCM suitable for certifying software requirements of outsourcing projects. Identification of these weaknesses helps in determining the direction in which the model needs to be improved.

This thesis proposes a flexible model (which can provide the core for a new version of LSPCM) which can be used for certifying software requirements of outsourcing projects. It also proposes a method for objective measurement of quality of the requirements, which means that the result of assessment will be more or less the same irrespective of the experience of the quality reviewer. The new model will be flexible in the sense that it can be

easily adjusted to certify requirements for a particular IT/Business domain. Also this new model automates most of the quality checks which are done manually in LSPCM, thus reducing the time required for certification significantly. Unlike LSPCM, it takes into account the availability of project-related documents for calculating certification level.

Acknowledgements

I would like to express my sincere gratitude to my project supervisors dr. Alexander Serebrenik and ir. Martijn Klabbers for their continuous support, stimulating suggestions and guidance throughout the duration of my thesis project and report writing. Without their encouragement and suggestions, this project might not have seen its completion. I would also like to thank prof. dr. Mark van den Brand for his valued inputs and advices throughout the project. I am greatly indebted to prof. dr. Jos Baeten, dr. Manohara M. Pai, and dr. Radhika M. Pai for considering me worthy for this great opportunity to gain and expand my knowledge through this program. Finally, I would like to thank my family, friends and colleagues at LaQuSo for their moral support.

Arpit Sharma
Eindhoven
20th August 2009

Table of Contents

List of Figures

List of Tables

1. Introduction

The quality of software requirements is a limiting factor on the success of software development projects. Poor quality of requirements alone can ruin any application development initiative. No matter how well-architected, well-constructed, or well-tested an application might be, it is essentially useless if it fails to meet software requirements. Defects in requirements are the source of the majority of defects that are identified during testing, and problems with requirements are among the top causes of project failure. There are three types of common requirements problems: incomplete or missed requirements, poorly managed requirements change, and inaccurate requirements.

Software requirements become more important in determining the success of a project in the context of outsourcing, as companies have to address several challenges which are specific to outsourcing [1, 2]. Due to communication challenges and cross-cultural issues, companies are unable to express requirements in such a way that vendor could correctly interpret them and implement in the project. Also different development practices and skill levels may result in wrong interpretation of software requirements.

Certification of software artifacts that are created during different stages of software development offers companies more certainty and confidence about software. An independent third party can be involved in verifying the quality of the software system. Third party assessors can produce an objective and complete judgment about the software quality and hand out certificates based on a set of predefined rules to judge the software quality. This third party assessment required for certification points out the defects in an early stage of software development, which makes companies more certain about software development, quality of software, planning and costs of their software end product.

The Laboratory for Quality Software (LaQuSo) is involved in verification and validation of software systems and their intermediate artifacts. LaQuSo has a product certification model called LaQuSo Software Product Certification Model (LSPCM) which can be used to certify software requirements along with various other software artifacts [3]. It can help outsourcing partners, the outsourcing company as well as vendor company, to convince the other party that requirements for the project are acceptable. This could help in preventing poor quality of requirements. The quality assessors will point out the defects present in the requirements and also provide suggestions to improve the quality of requirements. Although the model has a good set of rules to evaluate the quality of software requirements, it offers room for improvement.

One of the major weaknesses of this model is that it does not consider/discuss the issues which are very important for outsourcing and thus quality demands imposed by them are either absent in the model or their relationship with the outsourcing issues has not been

identified (e.g., how time to market affects prioritization of software requirements). This poses a threat in long run when the model is applied to outsourcing projects it would not be able to capture the defects in requirements. Also since time to market is very important for outsourcing projects, these issues need to be prioritized depending on the business domain. As the time available for verifying and improving the quality of requirements is short, certification needs to be given based on the prioritized quality demands.

Another major weakness of LSPCM is that it has lot of room for subjective interpretation, which means that certification level may change depending on the person who is assessing the quality of requirements. This uncertainty can be more problematic in outsourcing projects, as the requirements need to be developed with much greater accuracy and detail because of geographical and temporal separation between the outsourcing company and the vendor company. Individual team members of vendor company do not have the luxury of getting the answers and clarifications as quickly and easily as if their colleagues/customers were sitting just a few seats away.

Amount of time required for quality assessment using LSPCM is also high because most of the conformance quality checks are done manually.

The model also has some minor weaknesses (e.g., availability of project-related documents is not taken into account in calculating overall certification level) which are described in Section 4.3 of Chapter 4, where LSPCM has been discussed in detail.

In order to improve the model so that these weaknesses can be tackled, a number of research questions need to be answered which are explained below.

1.1 Research Questions

We address the following research questions:

1) What quality checks need to be added to the model so that it becomes suitable for certifying software requirements for outsourcing projects?

2) Is it possible to make the model sensitive to IT/Business domains?

3) Is it possible and if so, how to measure the quality of the software requirements objectively using LSPCM, leaving no room for subjective interpretation?

4) Can the time required for assessing the quality of requirements be reduced, by automating most of the quality checks that are performed manually in LSPCM?

5) Is it possible to provide the overall certification level in such a way that the maturity of software artifacts is better reflected?

1.2 Approach Used

To find out the answer to these questions and to improve the model following steps need to be carried out:

1) Identify those aspects of software development that are important in the context of outsourcing and do influence software requirements.
2) Make a categorization of domains in IT based on the aspects identified in first step.
3) Extract general outsourcing framework from management-oriented literature on outsourcing.
4) Identify and list the quality demands imposed by the aspects identified in step 1.
5) Prioritize the quality demands according to their importance for all the domains identified in step 2.
6) Construct a new quality model.
7) Provide quality metrics for each quality demand, so that it can be measured objectively.
8) Explain the assessment process to be followed in order to hand out certification for a project.
9) Determine the new criteria achievement levels and overall certification levels.
10) Try to automate the quality checks by using requirements measurement tool.
11) Validate the newly proposed model by applying it to a case study.
12) Provide conclusions and recommendations.

1.3 Report Outline

The rest of this thesis is organized as follows. Chapter 2 gives an overview of requirements engineering. Chapter 3 focuses on software outsourcing and the aspects that influence software requirements in the context of outsourcing. Chapter 4 briefly describes LSPCM and its weaknesses. Chapter 5 presents the quality model and explains the assessment process.

Chapter 6 gives a brief outline of how requirements measurement tools can be used to reduce the time required for quality assessment. Chapter 7 provides the validation results obtained by applying the model to a case study. Chapter 8 contains conclusions and recommendations.

2. Overview of Software Requirements

2.1 Introduction

The *IEEE Standard Glossary of Software Engineering Terminology* [4] defines a requirement as

1. A condition or capability needed by a user to solve a problem or achieve an objective.
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
3. A documented representation of a condition or capability as in 1 or 2.

In simple words, a requirement is a statement of need, something that some class of user/customer or other stakeholder wants. These stakeholders include developers, testers, project managers, documentation writers, legal staff, sales, marketing, manufacturing, help desk and other people who will have to work with the product and its customers. Requirements can be broadly categorized as business requirements, functional requirements, non-functional requirements, external interface requirements, constraints and behavioral properties. Requirements are crucial to every project and quality of requirements play a major role in determining the success or failure of a project.

2.2 Requirements Engineering

There are many definitions of requirements engineering (RE); however, they all share the idea that *requirements engineering* involves finding out what people want from a computer system, and understanding what their needs mean in terms of design. RE is closely related to software engineering, which focuses more on the process of designing the system that user wants. Kotonya and Sommerville define requirements engineering as "The systematic process of eliciting, understanding, analyzing, and documenting the requirements" [5]. Dubois, Hagelstein, and Rifaut used the term to refer to the part of development life cycle in

which the needs and requirements for the user community are investigated and then abstracted to create formal specifications [6].

The five specific steps in software requirements engineering are: requirements elicitation, requirements analysis, requirements specification, requirements validation and requirements management as shown in Figure 2-1.

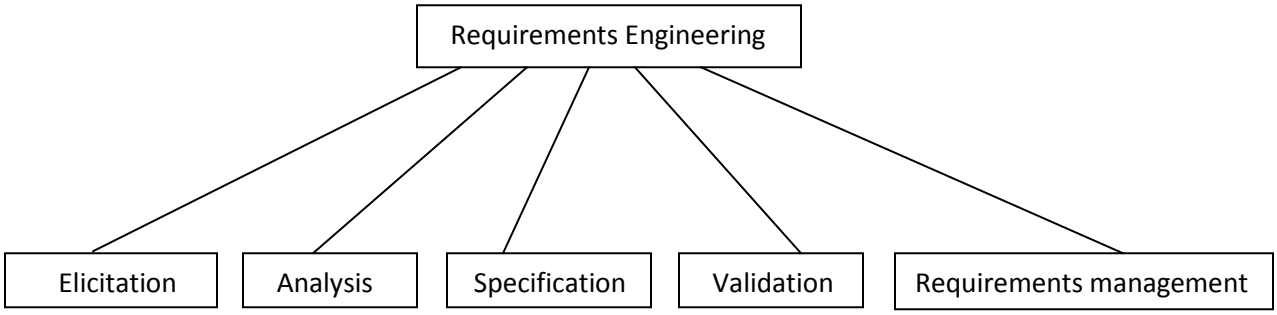

Figure 2-1 Steps of requirements engineering domain

These steps are explained below:

- **Requirements elicitation** is the first step in building an understanding of the problem the software is required to solve. It involves identifying the stakeholders, business objectives, operational and organizational environment. Once these sources have been identified, the requirements engineer can start eliciting requirements from them. Techniques that are used to elicit the right information includes, interviews with stakeholders, developing use cases, developing prototypes for clarifying unclear requirements, conducting facilitated meetings to refine ideas and identify conflicting requirements, and observation where software engineers learn about user tasks by immersing themselves in the environment and observing how users interact with their software and with each other.

- **Requirements analysis** involves refining the requirements to ensure that all stakeholders understand them and examining then in detail for errors, inconsistencies, ambiguities, and omissions. Analysis includes decomposing high-level requirements into details, understanding the risks associated with implementing each requirement, creating a data dictionary for consistent data definitions, prioritizing the requirements to resolve conflicts and allocating requirements to subsystems.

- **Requirements specification** involves producing a document, or its electronic equivalent called software requirements specification which can be systematically reviewed, evaluated and approved. It forms the basis for agreement between customer and contractor on what the software product is supposed to do. It provides a realistic basis for

estimating product costs, risks and schedules. It also forms the basis for testing the functionality and software enhancement. Software requirements are often written in natural language, supplemented with formal or semi-formal descriptions. IEEE has a standard, IEEE Std 830, for the production and content of the software requirements specification.

- **Requirements validation** - The requirements need to be validated to ensure that the requirements engineer has understood the requirements, and it is also important to verify that a requirements document conforms to company standards, and that it is satisfies various important quality demands like consistency and completeness. In order to measure the quality of requirements objectively, requirements quality metrics need to be created and applied to the requirements document. One of the goals of this project is to propose such metrics and to automatically apply them to requirements.

- **Requirements management** - Change is inevitable. Knowing how to handle changes and requests for changes is vital to delivering the right software product on time. Properly estimating the impact of a requested change and communicating that impact is vital to success of the project. Tracking the status of each requirement as it moves through development and system testing provides insight into overall project status.

2.3 Why Quality of Requirements is Important

The quality of any system depends on the quality of the raw materials fed into it, poor quality requirements can never result in a software system that is acceptable by the customer. However well the system may appear to work at first, if it is not the system that users want or need then it is useless. Figure 2-2 shows how the quality of requirements impacts the system that is developed [7].

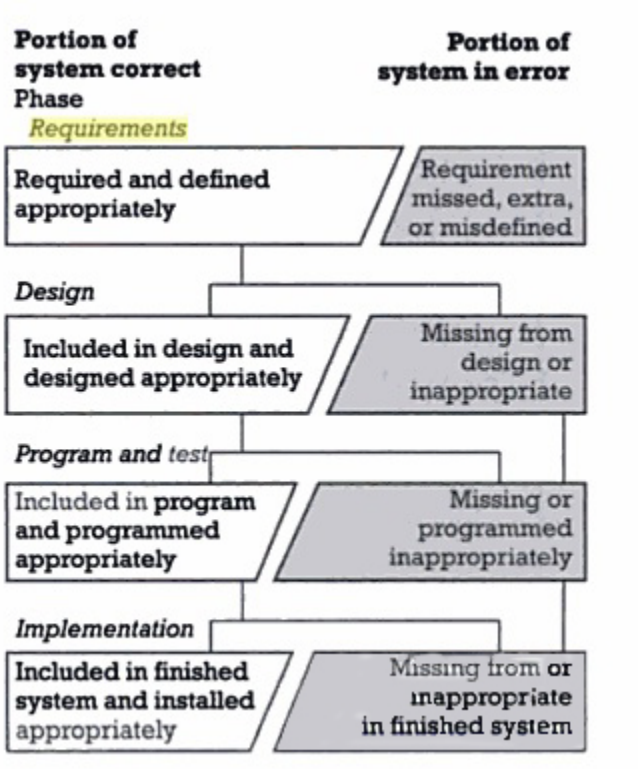

Figure 2-2 Error sources by phase

Of course the quality of requirements is not the only factor that influences the quality of a system. Quality of the system engineers, quality of the process, and available resources also play an important role.

The development of the system starts by defining the requirements, which are obtained by applying the steps of requirements engineering iteratively. Some of these requirements may contain errors such as ambiguous and redundant information.

When these requirements are translated into design, some of the appropriate requirements are lost in translation. It is similar to loss of meaning when translating from one language to another. Due to this reason the portion of the system in error is increased. Similarly some of the appropriate design is translated into appropriate program code, and some portion is lost during the translation. Thus the portion of the system in error keeps on increasing as we go through different phases of development.

Change in requirements during the development may also result in a system which is not acceptable, if the changes have not been reflected in the requirements document and subsequent phases of development.

Once an error appears in some part of the design or some other software artifact due to problems in artifacts of the previous phase, it is not easy to detect it, as those who are involved in that phase have little information or access to details of prior phase of development. For example an engineer involved in coding the software will have little information about the business objectives and software requirements, resulting in defect remaining unnoticed.

Since requirements are easy to modify and relatively cheap to change when they are being written, errors should be detected and removed in the requirements phase itself. The longer an error remains undetected during development the more expensive it is to fix. According to Boehm 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.

2.4 Benefits of High Quality Requirements

Good requirements practices result in high quality requirements which provide numerous benefits [8, 9]. Effective user involvement in establishing the requirements reduces the chance that user will reject the new system upon delivery. It will also help development team to emphasize user tasks and avoid writing the code for unnecessary features. Well-understood and well-documented requirements will help the development team to estimate the effort and resources required to execute the project. Unambiguous and testable requirements enable the testers to develop accurate test cases and test procedures to verify the functionality of the software. An effective change-control process which is a part of good requirements practices will ensure that change in requirements will not adversely impact the system.

3. Software Outsourcing

3.1 Introduction

Information technology has become one of the fastest-growing, fastest-changing markets in the world. The demand to produce new or specialized software for specialized markets, business, government or domestic use is ever increasing. As the scale and complexity of the marketplace continues to grow, many of the opportunities are turning into threats. Far from maintaining the pace of the change, the industry is seeing customer needs outstripping its ability to deliver. Indeed, the need of outsourcing is driven by the inability of organizations to develop software product fast enough or change their infrastructures or staff expertise to keep up with the changes in the information technology world and the demands imposed on them.

Software outsourcing could be defined as contracting out the development, planning, management, training, maintenance, or operation of software services, skills, products or applications. Outsourcing has been driven by the following factors [10, 11]:

- cost savings,
- faster time-to-market,
- timely access to highly qualified technical talent,
- increased competitive ability,
- need of non-core expertise,
- focus on core competencies,
- proximity to the international market,
- opportunity to improve the quality of software product.

Outsourcing can be domestic or international. Domestic outsourcing takes place locally within one country, when the outsourcing service customer and the vendor/provider are collocated within the same geographic location. International outsourcing, also called off-shoring, refers to outsourcing activities done between two or more countries. In both cases the assignment can be transferred from one group to another group either within the same company or between separate companies. Throughout this thesis report outsourcing will refer to international outsourcing or offshoring.

3.2 Challenges Involved

Some of the major challenges involved in software outsourcing are as follows [1, 2, 11, 12]:

- **Social and cultural boundaries** – Social and cultural distance can result in a fundamental difference in opinion about the nature of the software development process. It can lead to misunderstandings and non-native speakers struggling to follow the technical discussions and meetings. Lack of familiarity with the remotely located colleagues can result in a lack of tameness and a reduced sense of trust. Certain complex aspects of a society can be easily misunderstood by outsiders and are sometimes so subtle that they may go unnoticed altogether.

- **Temporal and geographical separation** – Lack of informal communication between two teams involved in software development is a major challenge, as written documentation is often inadequate when resolving misunderstandings, such as misunderstandings about requirements or changes in requirements specification. Geographical and temporal distances make it more difficult to initiate contact with colleagues at other locations. A lack of overlapping working hours can lead to delays in feedback, making the development process less effective.

- **Differences in development practices and technical skill levels** can also result in misunderstandings and conflicts/tensions between the two teams. For example most of the employees of a vendor company are new college graduates with no prior experience of software development, and thus not suitable for high-skill tasks of analysis and design.

3.3 Importance of Certification in Outsourcing: The Case of a Russian-Norwegian Software Outsourcing Project

In order to understand the importance of certification in software outsourcing, a case study is discussed briefly in this section [13]. The case analysis identified various types of problems faced by the companies involved in the project. Subsection 3.3.1 gives an overview of the project. Major problems which influenced the development adversely have been listed in Subsection 3.3.2. The role software certification could have played in this case and how it could have solved some of those problems is presented in Section 3.3.3.

3.3.1 Overview of the Project

ScanSys (the Norwegian vendor) is a company with 13000 employees located across Europe, headquartered in Finland. ScanSys' main business is in providing consultancy services on business systems to its customers. RussCo (the Russian contractor) with 110 employees and situated in St. Petersburg, provide offshore software development services. The project given to RussCo by ScanSys was called SalarySystem, a payroll management system used by nearly 11000 small and medium sized companies in Norway. The project involved reengineering SalarySystem from its existing expensive, licensed, and soon getting obsolete database, to the new, free of charge and more powerful MySQL platform.

3.3.2 Problems Faced During Development

As a result of various challenges involved in outsourcing which have been mentioned in Section 3.2, the development was adversely influenced and the project was nearly terminated. The major problems responsible for creating a near-breakdown situation were identified as follows:

1) SalarySystem was deeply connected to the complex and particular system of Norwegian tax and salary rules, which were annually updated. The actual business logic was understood by few people at ScanSys who were no longer in the organization. Also the code was not well documented and did not follow standard coding conventions with respect to comments, code structure, use of variable names etc. This resulted in lack of understanding of the system by Russian developers and thus system was specified incorrectly.

2) The documentation of the SalarySystem was to a large extent written in Norwegian, including the comments on the source code, class names, variable names, database fields and tables. As a result Russians received very limited understanding of the system requirements, and technical terminology could not be correctly interpreted. Also huge amounts of time were taken for translation.

3) No visit from a Norwegian side was made to Russia for shared understanding of the needs and project progress. The only method of communication used was e-mail, which did not satisfactorily answer the queries posed by Russian developers and sometimes took three days to receive a reply from Norway.

3.3.3 How Certification Could Have Helped

Certification of software artifacts by an independent third party could have helped the development in the case of SalarySystem. Certification is a check that the artifact fulfills a well defined set of requirements. In this case it is clear from the previous discussion that the business objectives were not clear, requirements were incomplete and could not provide the details about what needs to be done, test cases were not provided along with the requirements document and no effort was made by Norwegian side for translation of requirements document in English. A certification model could have been applied to these documents, which would detect the defects present in requirements, make sure that the document is structured and complies with company or industry standards. The certification will make sure that all the important documents have been provided and traceability links will indicate the omissions and extra features. Also issues related to language differences would be solved by if the certification model clearly states that the requirements should be written in a language understood by both the parties, and a glossary of domain specific technical terms/abbreviations/acronyms should be included with the document. Similarly code certification process will help in identifying the problems related to standard coding conventions, programming language errors, code maintenance and traceability to design. Also test plan documents can be checked against the rules in the model to make sure they are of high quality. Certification will be given only if the artifacts are high quality, which also help the outsourcing partners, to convince the other party that deliverables are acceptable.

3.4 Aspects of Software Development

In this section those aspects of software development are discussed, which influence software requirements either directly or indirectly and are relevant in the context of software outsourcing. These aspects are important as they will help in identifying some of the quality demands which should be included in the model and measured objectively, to make the model suitable for certifying requirements for outsourcing projects.

- **Time to market** – Time to market reduction is one of the most effective ways to increase market share and product revenue. It becomes an important factor determining the success of a product in market for those companies which are involved in developing software with the aim to make money from the use of the software. These companies are under constant pressure to launch their products as soon as possible to capture different market segments. If the companies do not have time or resources to deliver the project in the required time frame they outsource the software development to other companies.

If the time to market is short, requirements should be prioritized to make sure the product contains the most essential features. Requirements should be prioritized from the perspective of how valuable each one is to the success of the product in market.

When time to market is short, time spent on refining the requirements is limited. For example time spent on formal inspection methods would be limited and hence some minor defects in the requirements should be acceptable. One cannot expect requirements for every product feature to be expressed in detail, and lower quality of requirements for non essential features should be acceptable.

- **External changes** – A key problem in outsourcing software development is the changing requirements. No matter how through the requirements specification has been setup, the requirements for any complex product will change, during the process of implementing the system. This change can be due to many reasons, including change in economic and legal policies, project team interpreting requirements different than intended by stakeholder, forgotten requirements pop up and technological developments.

  Due to change in requirements the requirements document itself becomes inconsistent, which may result in a system that could not be implemented, misinterpretations or false assumptions by developers or delivery of a system that will not be accepted by the client. Therefore change in requirements or business logic should be reflected consistently in the requirements document and flow through the architecture and design rather than implementing them directly into code.

  Communicating the reason and impact of changes in requirements to the subcontractor becomes a difficult task due to communication barriers and cultural differences. As a result the chances that the final product will not meet all the requirements are high.

- **Impact of failure** – Certain software systems can be mission critical, as the failure to provide the desired functionality may result in human and property loss. For such systems the desired functionality, operating environment, real time constraints on the system, should be clearly understood by the vendor company developing the system. Also the potential risks involved and the plan for recovery from failures without causing any damage should be identified and analyzed. Since there is no chance for errors, execution of development activities without negligence and omissions is must. When such systems are developed outhouse. The vendor company developing the software might not understand the complexity involved and the sense of responsibility for the actions taken by it. This can happen because of the communication challenges or low degree of monitoring and cultural differences

involved in the outsourcing. The quality of requirements become very important and every single detail about the software functionalities and exception situations should be included in the requirements document. These details should be written unambiguously and consistently, to make sure they are not wrongly interpreted. Also formal verification of behavioral properties and constraint requirements can be done to identify the errors.

- **Constraints** – Constraints are the design and implementation restrictions imposed on the choices available to the developer for some legitimate reason. In other words constraints are effectively global requirements, such as limited development resources or a decision by senior management that restrict the way system is developed. Constraints can be technical, political, economic or environmental and pertain to your project resources, schedule, target environment or to the system itself. These constraints should be satisfied by the final system. Vendor company may need to recreate the operating environment to check if the product works under the given constraints, and thus they must be clear to the vendor company. Also specifying the reason for the presence of constraint will help the vendor company in understanding its importance.

- **Cross cultural issues** – Two companies involved in the partnership may have different cultural backgrounds, for example different languages, values, norms, ways of perception [12]. These differences may result in wrong interpretation or misunderstanding of business logic. The requirements in the document may be vague when taken out of the context of the culture in which the software will be used. If the developer is not familiar with the business cultural rules it would be impossible to correctly implement the requirements. We have already seen an example of impact of cross-cultural issues on requirements in Section 3.3, where requirements were written in Norwegian. Outsourcing team may use a metaphor in the diagrams in the requirements document that the vendor team is not familiar with, or find confusing or offensive. The problem even becomes magnified more when we are talking about the software products which require huge amount of user interaction. Culture preferences and biases (i.e., colors, text vs. graphics, spatial orientation) effect the way interface is designed. For example red color represents happiness in China, anger in Japan, aristocracy in France and danger/stop in United States. In order to overcome these problems there is a need to develop culturally-aware requirements specification which means more than just declaring who will use the system.

3.5 IT Domain Categorization

Based on the aspects of software development identified in Section 3.4, we can distinguish the software products from each other and also put software products in different categories as shown in the Table 3-1 below. The aim of this Section is not to present a complete and detailed categorization of all software products, but to show that these aspects can be used to distinguish one group of software products from another.

	Time to Market	External Changes	Impact of Failure	Hardware Constraints	Cross Cultural Issues
Military Software		?	✓	✓	✓
Governmental Software		✓			✓
Web-based Retail System	✓	?			✓
Medical Software			✓	✓	
Gaming Software	✓			✓	✓

Table 3-1 IT domain categorization

In the Table 3-1 only those aspects which are relevant for a particular software product have been marked. Since within a category there can be exceptions, only those aspects which are very important for a particular category as a whole are included. For example gaming software which simulates the cockpit environment can be life threatening in the sense that if some requirements have been missed in the implementation, it may result in pilots not getting the complete information about the controls inside the cockpit of an airplane.

A brief description of these software products/software categories is as follows [14]:

- **Military software** – Refers to software produced for a uniformed military service such as the nation's air force, army or navy. An example of this category software is pattern-recognition software used to guide cruise missiles. Since these systems operate in real-time they have stringent hardware constraints and also failure can be life-threatening.

- **Governmental software** – Refers to the software produced for the public sector organizations. An example of this category software is municipal billing and tax

collection software. The business logic of this software may change with time due to new tax rules and policies.

- **Web-based retail system** – Refers to the software used for online retail and payment processing. It allows clients to purchase products via internet. In order to capture the market share retail stores want the software to be deployed as soon as possible.

- **Medical software** – Refers to the software used to control medical devices that monitor patients, diagnose diseases, collect data and present it in a meaningful way. Example of medical software includes software for CAT scanners, pacemakers, and infusion pumps. Since medical software operates in a well defined environment without much interaction with the user, thus cross cultural issues are not very relevant.

- **Gaming software** – Different types of computer games such as arcade-style, first person shooting, simulation games (flight simulators, war simulators and business simulators) can be used for recreation and learning purposes. Software products in this category usually have short time to market and tight hardware interface constraints.

3.6 Outsourcing Life Cycle

Companies spend high amounts of time and money on exploring the ways to execute numerous tasks involved in different phases of complex software outsourcing process, but still fail in this due to lack of systematic and detailed framework.

This section describes the general outsourcing framework/life cycle which can be used for outsourcing of software production [15]. This framework can help in proper planning, control and continuous improvement of the outsourcing relationship.

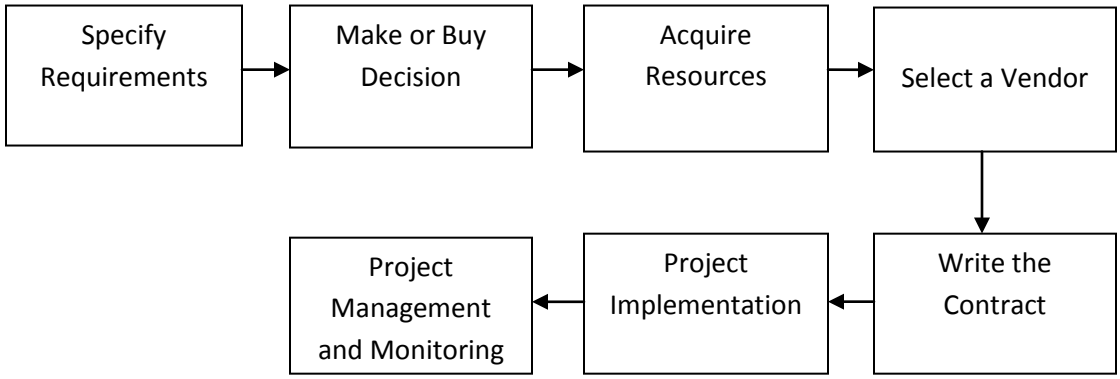

Figure 3-1 Phases of general outsourcing framework

Various phases involved in the general outsourcing framework are shown in Figure 3-1 and explained below:

1) **Specify requirements** – Requirements document is a part of legal contract between the outsourcing company and vendor in the contract writing phase, so it should clearly state what needs to be done. But in the first phase the requirements may not be at a very detailed level, but they should be able to help the outsourcing company in rough cost and time estimation.

2) **Make or buy decision** – Before taking the decision to outsource, an internal outsourcing team should be formed which would be involved in identifying the core competences, determining software components that could be outsourced, weighing both costs and benefits. The decision to outsource should be taken with full awareness of potential problems and risks involved.

3) **Acquire resources** – Outsourcing company should acquire sufficient resources to complete the project, which includes budgets for the initial proposal phase, vendor payment, and management of the project on the outsourcing company's side. Since a lot of activities may be involved in the proposal phase (e.g., communication with the potential vendors or setting up a team to evaluate the proposals) required budgets must be acquired.

4) **Select a vendor** – Various activities involved in this phase pertain to identifying potential vendors based on maturity levels, domain expertise, and other criteria such as political and social environment of the countries where projects can be outsourced. This phase also involves creating and issuing request for proposal, which includes requirements document, statement of work, documentation list, cost and schedule estimates. Vendors should be informed about the evaluation criteria such as technical capability, design

approach, management capability and proposal preparation guidelines. Proposals obtained from different vendors should be compared and evaluated to select a winner.

5) **Write the contract** – Details of management requirements, technical requirements, warranties, patents, intellectual property issues, contract termination, payment and other important issues should be included in the contract. In this phase the requirements document should describe the system in sufficient detail. Also there should be an agreement between two parties about how change in requirements will be handled.

6) **Project implementation** – This phase involves building an efficient implementation team, developing a clear implementation plan, determining the development methodology, training members of both the parties, carrying out the development activities, and jointly reviewing the milestones.

7) **Project management and monitoring –** Since it is not possible to monitor the progress of an outsourced project just by walking around and talking to project team members, project tracking indicators defined in the contract should be strictly applied to the project. This phase involves creating the management structure, communication and sharing of knowledge, developing measures to evaluate the performance, monitoring the vendor's progress and realigning the contract if required.

4. The LaQuSo Software Product Certification Model (LSPCM)

LaQuSo (Laboratory for Quality Software) has developed a model for software product certification which is called the LaQuSo Software Product Certification Model (LSPCM) [16].

4.1 Overview of LSPCM

LSPCM can be used to analyze and certify not only the final software product which is the working system, but also intermediate deliverables such as user requirements, high-level design and detailed design. These software artifacts can be put into different categories called Product Areas.

The model consists of six Product Areas which are as follows:

- The context description *(CD)*

- The user requirements *(UR)*
- The high-level design *(HD)*
- The detailed design *(DD)*
- The implementation *(IMP)*
- The tests *(TST)*

This thesis focuses on the second Product Area, the user requirements and therefore this and subsequent sections will explain LSPCM from requirements point of view. The user requirement area can be further divided into subparts, called *elements*. These elements can be separate artifacts, a chapter within a document, or different parts of a larger model. LSPCM divides user requirements into four elements: use cases or functional requirements, non-functional requirements, behavioral properties and objects. The concepts of the certification model are shown in Figure 4-1.

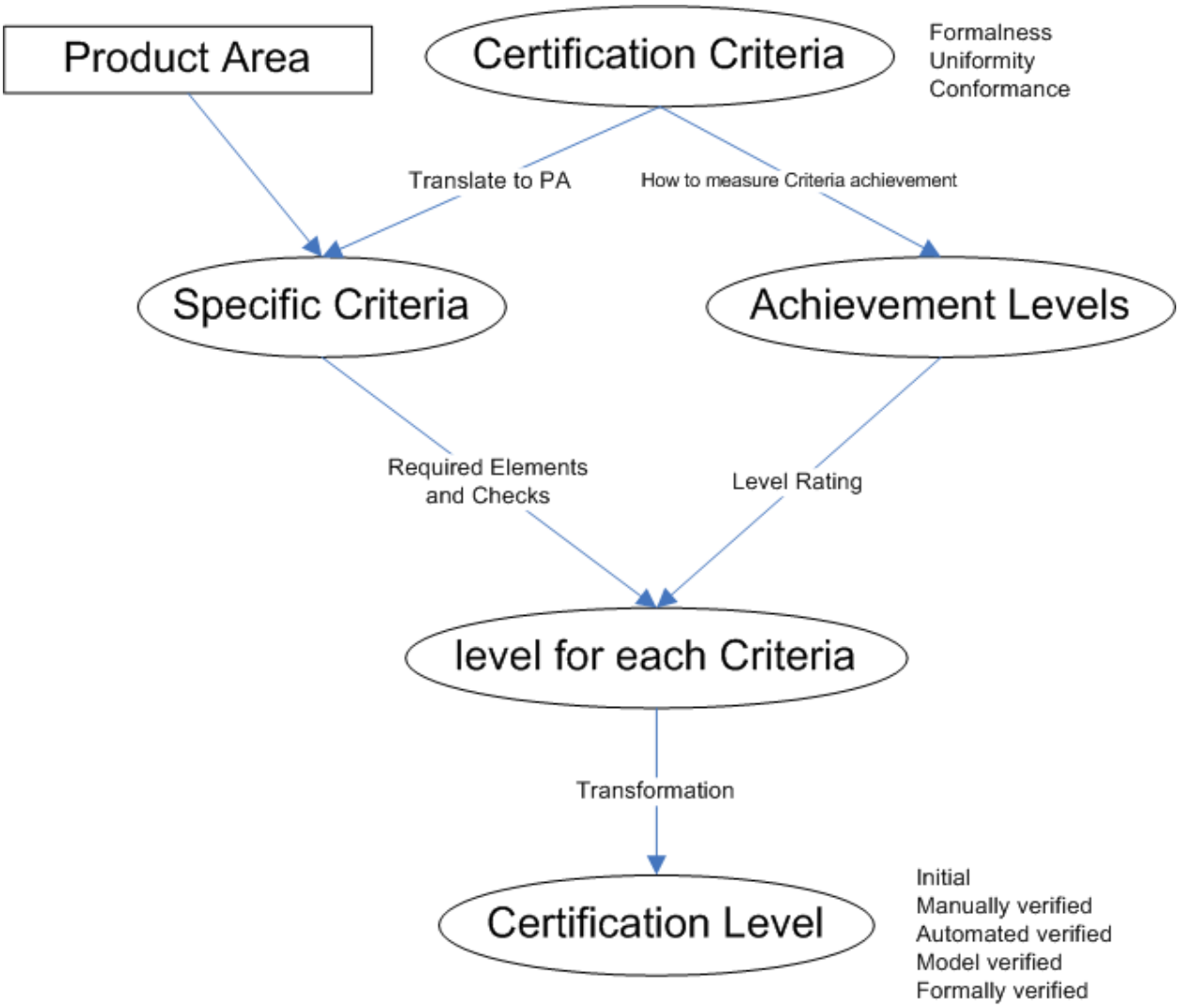

Figure 4-1 Concepts of the Certification Model

In order to analyze the artifacts LSPCM defines three Certification Criteria *(CC)* for all Product Areas:

[CC1] **Formalness –** All required elements in the Product Area should be present and as many (derived) formal elements should be present as possible. Functional requirements is an example of the required elements and relational diagram of data/object model formally specify the requirements.

[CC2] **Uniformity** – The style of the elements in the Product Area should be standardized.

[CC3] **Conformance** – All elements should conform to the property that is subject of the certification.

For each of these Certification Criteria different *Achievement Levels* can be established as shown in Table 4-1.

CC1	**Formalness**
0	Some required elements are missing
1	All required elements are present
2	Semi-formal elements have been added
3	Formal elements have been added
CC2	**Uniformity**
0	No standardization
1	Within the product
2	Style complies to a company standard
3	Style complies to an industry standard
CC3	**Conformance**
0	Faults are detected
1	Manual review/testing has not detected any faults
2	Automated approach has not detected any faults
3	Formal verification has not detected any faults

Table 4-1 Certification Criteria Achievement Levels

For each Product Area there is a different set of Specific Criteria which indicates the elements or checks that are required for a certain Certification Criteria Achievement Level.

A brief description of the Specific Criteria that can be applied to User Requirements is as follows:

[SC1] **Complete** – The requirements are as detailed and as formal as possible.

[SC1.1] **Required Elements**

- Functional requirements.
- Non-functional requirements.
- Glossary.

[SC1.2] **Semi-formal Elements**

- Data dictionary or object model.
- Use-cases (with scenarios).
- Flowchart of processes.
- Behavioral properties.

[SC1.3] **Formal Elements**

- Relational diagram of data/object model.
- Process models of use-case scenarios.
- Behavioral properties specification.

[SC2] **Uniform** – The style of the requirements description complies with the standards in requirements engineering.

[SC2.1] **Uniformity within the Project**

- Elements and documents of the same type have the same style.

[SC2.2] **Compliance with Company Standards**.

- All elements and documents comply with company standards.

[SC2.3] **Compliance with Industry Standards**.

- ERD diagram for object/data model.
- UML diagrams for use-cases.

[SC3a] **Correct** – Each element in the requirements is described in a correct way.

[SC3a.1] **Internal Correctness**

- No two requirements or use-cases contradict each other.
- No requirement is ambiguous.
- Functional requirements specify what, not how (no technical solutions).
- Each requirement is testable.
- Each requirement is uniquely identified.
- Each requirement is atomic.
- The definitions in the glossary are non-cyclic.
- Use-case diagrams correspond to use-case text.

[SC3a.2] **Automated Correctness Checks**

- Requirements are stored in a requirements management tool which uniquely identifies them.

[SC3a.3] **Formally Verified Correctness**

- The use-case scenario models are correct workflows (no deadlocks or dead tasks).
- The use-case scenario models are mutually consistent.
- The data model diagram is in normal form.

[SC3b] **Consistent** – The relations between the elements in the requirements description and with the context description are correct and consistent.

[SC3b.1] **External Consistency**

- Each ambiguous or unclear term from the requirements is contained in the glossary.
- The use-cases or functional requirements detail the environment description.
- Each object is mentioned in the requirements and all objects mentioned in the requirements are contained in the object model.
- For each object create, read, update and delete operations are covered in the user requirements or not applicable.
- The requirements do not contradict the behavioral properties.

- The use-case or functional requirements do not render the non-functional requirements impossible.

[SC3b.2] **Automated Consistency Checks**

- Requirements and glossary/objects are stored in a requirements management tool which shows the relations between requirements, scenarios, actors and objects.

[SC3b.3] **Formally Verified Consistency**

- The use-case scenario models are compliant with the behavioral properties.
- The use-case scenario models are compliant with the non-functional requirements.
- The requirements description (process model and behavioral properties) complies with the environment description (process model) from the context description.

4.2 Certification Levels

To certify the requirements document the model has five certification levels. The first one is only used for creating a baseline that the certification analysis can start. These overall certification levels can be calculated from the levels achieved for the three certification criteria as follows:

1. **Initial**

   $CC1 \geq 1$ and $CC2 \geq 1$ and $CC3 = 0$

   Each of the required elements is present in the product and the elements are uniform.

2. **Manually verified**

   $CC1 \geq 1$ and $CC2 \geq 1$ and $CC3 = 1$

   All elements, relationships and properties have been manually verified.

3. **Automated verified**

   $CC1 \geq 1$ and $CC2 \geq 1$ and $CC3 = 2$

   All elements, relationships and properties have been verified with tool support.

4. **Model verified**

CC1 ≥ 1 and CC2 ≥ 1 and CC3 = 3

All elements, relationship and properties have been verified with mathematically-based methods wherever possible, or the most rigorous method otherwise.

5. **Formally verified**

   CC1 ≥ 1 and CC2 ≥ 1 and CC3 = 3 and 'Model == Input'

   Model verified where it is proven that the results of the mathematically-based methods are true for the actual input (and not only for an abstract model of it).

4.3 Weaknesses of LSPCM

LSPCM is a rule-based model which only allows the third party to assess the product quality and is not concerned with management and development processes. The model can be used to identify the defects in the requirements document, but there are some weaknesses of the model which need to be improved to make the model more concrete. These weak points have been briefly described below in the order of their importance.

- LSPCM does not consider the aspects of software development which are important for outsourcing, and thus fails to establish the relationship between these aspects and quality demands. Some important quality checks imposed by these aspects are also missing in the model. This can be a threat in long run because when the model is applied to outsourcing projects it might fail to detect some defects which appear only in case of outsourcing projects. For example, the model does not specify any rule to check the quality of requirements in presence of cross-cultural issues common in the outsourcing process.

- The model is not sensitive to IT domains in the sense that it does not prioritize the quality demands according to their importance for a particular domain. As we have seen in Section 3.5, certain aspects are more important than others for every IT domain, and thus, certification should be based on the prioritization of quality demands. If a requirements document satisfies all the high and medium priority quality demands the certification level should be high. For example some software products have short time to market and therefore all the features must be prioritized according to their importance in determining the success of the product, and this should be reflected in certification as well.

- The overall certification level is calculated based on the type of verification performed on the requirements document. But in actual practice very few requirement

documents are provided with formal elements and for most of the products formal verification of requirements is not required. It makes more sense to certify the requirements document based on the prioritization of quality demands and formally verify the requirements only for those products for which it is important. For such products formal verification of high and medium quality demands must be done to achieve a high overall certification level.

- It does not give weight to availability of project-related documents in calculating the overall certification level. For example a requirements document provided with test plan document and business requirements document can be checked more thoroughly. Traceability matrices between different elements can be compiled when these documents are present, and thus more in depth analysis can be carried out. This availability of documents should be reflected in overall certification level.

- LSPCM provides room for subjective interpretation, i.e., overall certification level may change depending on the person assessing the quality of requirements. For outsourcing projects the subjectivity can become more problematic, as the requirements need to be developed with much greater accuracy and detail because individual team members of vendor company do not have the luxury of getting their doubts and queries cleared as quickly and easily as if their colleagues/customers were sitting just a few seats away. Therefore there is a need to objectively measure the quality of requirements and determine boundary values.

- The quality checks for required elements are performed manually in LSPCM which consumes significant amount of time. In order to reduce the amount of time required for assessing the quality of requirements, these quality checks need to be automated.

5. Quality Model

The case study presented in Section 3.3 showed that the development of high quality requirements is a must for development of a high quality software product. The software product based on inaccurate requirements will be unsatisfactory. This chapter proposes a new quality model (which can provide the core for a new version of LSPCM) which can be used to overcome the weaknesses of LSPCM explained in Section 4.3. Section 5.1 briefly explains the quality demands and quality metrics. Section 5.2 describes the new quality model. A brief description of requirements quality metrics is provided in Section 5.3. Section 5.4 presents the assessment process. Section 5.5 discusses the new criteria specific achievement levels and overall certification levels.

5.1 Quality Demands and Quality Metrics

Requirements quality demands can be defined as the properties/characteristics that the individual requirements or requirements document as a whole should exhibit to make sure that the requirements are of high quality. In other words, these demands are requirements for requirements. Quality demands can help in identifying the defects in requirements document which should be corrected before development is carried out [17].

Measurement of quality demands may involve subjective interpretation which means we can never say something about the quality of requirements with absolute certainty. For example, the quality demand can be unambiguity which says that every requirement should only have a single meaning. In order to show how many of the requirements can be termed as ambiguous there should be some method to objectively detect the presence of ambiguities because otherwise it is totally dependent on how quality assessor interprets it.

Requirements quality metrics are used for objective measurement of quality demands [18]. One or more quality metrics can be associated with each quality demand to provide a quantitative assessment of the extent to which the requirements document possesses that quality demand. Quality metrics aim at eliminating subjective interpretation from the quality assessment of requirements.

5.2 Proposed Model

In order to overcome the weaknesses of LSPCM, a new quality model has been proposed in this section. The model has three quality criteria: availability, uniformity, and conformance. Each quality criteria has one or more quality rules/quality demands. Quality rules check the

presence of certain elements/documents and compliance of these elements/documents with company and industry standards. These quality criteria correspond to Specific Criteria for the User Requirements Product Area in LSPCM. One or more quality metrics will be used for objective measurement of each quality demand. The core concepts of this model are shown in Figure 5-1.

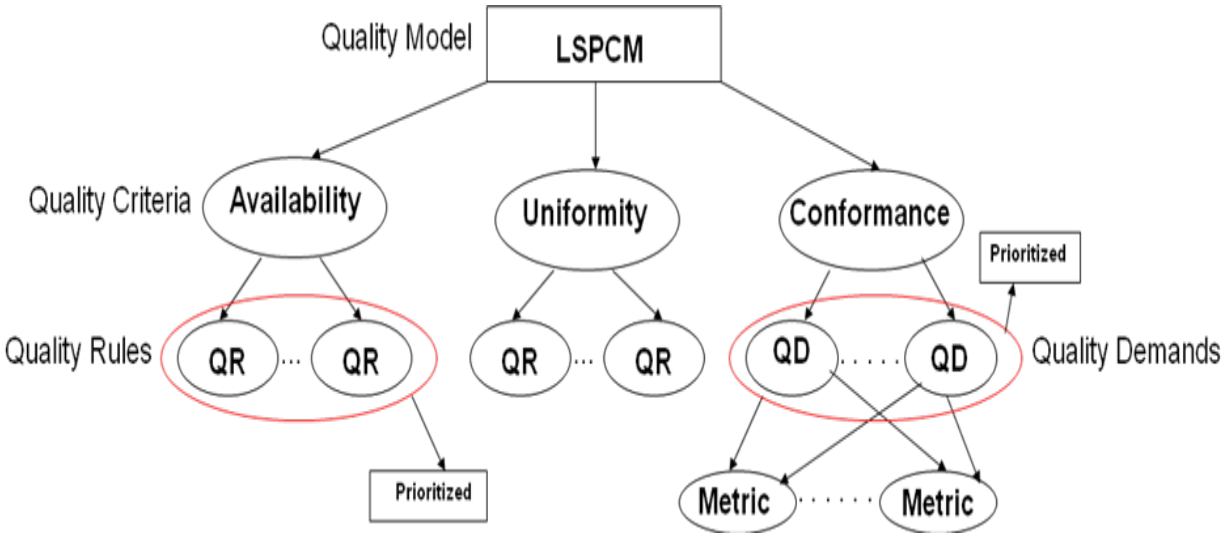

Figure 5-1 Core concepts of new model

All the quality criteria and their quality rules/quality demands (for software requirements) are explained below:

[SC1] **Availability –** It checks the presence of software requirement elements and other project-related documents. By identifying the aspects of software development which are important for the project and discussing the importance of project properties with the outsourcing party, these elements and documents can be prioritized. For example, response time and security can be very important for military software products and, thus, these non-functional requirements should be given high priority, whereas use-case diagrams can be given a comparatively low priority. There can be exceptions for every category and therefore, prioritization should be strictly based on the inputs from customers or outsourcing party.

The different elements of software requirements and project-related documents are explained below:

- *Functional requirements* – These are statements of services that the system should provide, how the system should react to particular inputs and how the system should

behave in particular situations. Functional requirements find a home in the software requirement specification document.

- *Non-functional requirements* - They are constraints on the services or functions offered by the system such as timing constraints. Non-functional requirements find a home in software requirement specification document.

- *Glossary* – is used to explain all the terms that are specific to the project or application domain. It also explains acronyms, abbreviations or other project or company specific shorthands. Glossary is usually included in the software requirement specification document.

- *Use-cases* - Use cases describe how external entities will use the system. These external entities can be either humans or other systems and are referred to as actors in UML terminology. The description emphasizes the users' view of the system and the interaction between the users and the system. They are usually in the form of a diagram, along with a textual description of the interaction taking place. Use case description can be included in the software requirement specification document.

- *Business requirements* - describe why the organization is undertaking the project. They state the benefits that the developing organization or its customers expect to receive from the product and are usually recorded in a vision and scope document. Business use cases, stakeholder information and main processes in the target system should be included in the business requirements. If business requirements are provided, traceability matrix can be compiled between software and business requirements to help in identifying the missing and extra requirements.

- *Interface specification* - These include interfaces to other software components, to hardware devices, and to human users, as well as communication interfaces and protocols use to exchange information with other systems.

- *Behavioral properties* - These are constraints which are specified in terms of formally analyzable natural language properties. If important for the project they can be included in the software requirement specification document.

- *Constraints* – They are the design and implementation restrictions imposed on the choices available to the developer for some legitimate reason.

- *Test cases* - A test case is usually a single step, or occasionally a sequence of steps, to test the correct behavior/functionalities, features of an application. Test cases are written in test plan document. If test plan document is provided, traceability matrix

can be compiled between software requirements and test cases which help in identifying the missing and extra test cases.

Prioritization of these elements and project related documents can be done as shown in Table 5-1 below.

Element/Project Related Document	**Priority**
Functional Requirements	*** (High)
Non-functional Requirements	**D (High or Medium)
Glossary	** (Medium)
Use-case	*D (Low or Medium)
Business Requirements	*D (Low or Medium)
Interface Specification	DD (Low or Medium if applicable)
Behavioral Properties	D (Low if applicable)
Constraints	*D (Low or Medium)
Test Cases	*D (Low or Medium)

Table 5-1 Element/Document Prioritization

In the Table 5-1, D represents domain specific and thus the priority can be increased by one star for each D present in the prioritization column of a particular element/document. As mentioned earlier for some products certain elements/documents can be more important than other products resulting in change in priority of those elements/documents.
Three quality checks that need to be performed after prioritization of elements/documents are as follows:

- Are all the high priority elements present in the requirements document?
- Are all the medium priority elements and/or documents available for assessing the quality of requirements?
- Are all the low priority elements and/or documents available for assessing the quality of requirements?

[SC2] **Uniform** – The style of the requirements description complies with the standards in requirements engineering.

Uniformity within the project - Elements and documents of the same type have the same style. E.g., all use cases have the same format.

Compliance with Company Standards - All elements and documents comply with company standards. There can be templates for requirements or use cases within the company. There can be standard diagramming techniques or formal languages.

Compliance with Industry Standards – ER diagrams should be used for drawing object/data model and UML diagrams should be used for drawing use-cases.

[SC3] **Conformance** – This quality criteria consists of seven quality demands: Completeness, Consistency, Prioritization, Testability, Unambiguity, Modifiability and Understandability. These quality demands should be prioritized based on the inputs from the customer or outsourcing party and the aspects which are very relevant for the project. Criteria specific achievement level would be given depending on how many quality demand priority levels are satisfied by the requirements document. Each priority level will have one or more quality demands and the level is said to be satisfied if all the quality demands at that level are exhibited by the requirements document. Quality demands are objectively measured using quality metrics which are described in the next section.

A brief explanation of these quality demands is given as follows:

- *Completeness* – Checks the necessary information missing from the requirements document. This information can be detailed software requirements, identifier for requirements, identifier or caption for figures and tables, use-case descriptions, use-case names, links in traceability matrices, referenced objects, explanation of abbreviations/acronyms, links between actors and use-cases, units of measurement. Completeness can be further classified into intra-document and inter-document completeness, based on the document where information should be present. A software requirement with no identifier is an example of intra-document completeness as the missing information can be identified from the requirements document itself. On the other hand, a software requirement with no test case is an example of inter-document completeness as it involves both requirements document and test plan document to identify the requirement with no test case.

- *Consistency* – Compares the elements of requirements document with elements of other project related documents or documents outside the project boundary to identify

different kinds of errors/mistakes present in the requirements document. Based on the other document used for comparison, consistency can be further classified into intra-document and inter-document (project related or outside project scope) consistency. Different kinds of errors/mistakes that can be identified are as follows:

1) Intra-document
   - duplicate requirements,
   - two or more requirements with same identifier,
   - use-cases referencing themselves (loop).
2) Inter-document (project related)
   - reference to a wrong element/figure/table/document,
   - usage of different terms/acronyms/abbreviations having the same meaning,
   - reference to a section/element/figure/table/document which cannot be identified.
3) Inter-document (outside project scope)
   - spelling mistakes,
   - grammatical mistakes,
   - use of different language words,
   - semantic related problems (e.g., response time should be 3 kilograms).

Consistency also includes those characteristics dealing with the presence of semantic contradictions in the requirements document, but they are hard to detect and involve subjective interpretation. These defects cannot be detected using the new approach, but suggestions can be provided by the experts assessing the quality of requirements about how to remove them.

- *Unambiguity* – A requirement is unambiguous if and only if it can be subject to only one interpretation. If a requirement can be interpreted differently by users, developers and other stakeholders in the project, it is quite possible to build a system that is completely different from what the user had in mind. Different types of ambiguity are explained below:

1) Lexical ambiguity – is ambiguity based on a single word. In many cases, a single word in a language corresponds to more than one thought, for example, the noun bank (financial institution vs. the edge of a river). Words may also have more than one meaning through their unrelated use in more than one category of speech, for example, *can* (a container of food – noun vs. to be able to – verb). Based on these words, ambiguous sentences can be further classified as follows:

   - Optional sentences (e.g., possibly, eventually, if needed, if appropriate),
   - Subjective sentences (e.g., similar, having in mind, better),
   - Vague sentences (e.g., bad, good, fast, recent, close),
   - Weak sentences (e.g., can, could, may).

2) Syntactic ambiguity – sometimes an entire sentence has more than one different interpretation, even if none of the words are ambiguous. An example of syntactically ambiguous sentence is given below:

   - The system shall not remove faults and restore service.

3) Semantic ambiguity – is based on specifications in multiple directions, across a text. An example of semantically ambiguous sentence is as follows:

   - System shall do within 5 seconds.

- *Understandability* – requirements document should be fully understood when used for developing software product. The presence of multiple sentences and implicit sentences (a sentence with subject containing a demonstrative adjective such as these, those, that) makes the requirements document more difficult to be understood and read. The requirements document should not contain underspecified individual requirements with a word identifying a class of objects without a specifier for this class. To make the requirements document more understandable comments and examples/illustrative information should be included in the document wherever required.

- *Testability* – if requirements are testable, there is a high probability that they will, in fact, meet the user needs as well as simplify implementation. If test plan document is available, traceability matrix can be compiled between software requirements and test cases to identify the requirements that are not yet tested and unnecessary/out of scope test cases present in the test plan document.

- *Modifiability* – requirements document is modifiable if its structure and style are such that any necessary change to the requirements can be made easily, completely, and consistently. To make the requirements document more modifiable related requirements should be grouped together and a requirement should not be in more than one place in the document. The requirements document should also have a table of contents, and cross-references if necessary.

- *Prioritization* – an implementation priority should be assigned to each requirement, feature to indicate how essential it is to include it in a particular product release. Requirements should be prioritized from the perspective of how valuable each one is to the success of the product in market.

5.3 Requirements Quality Metrics

This section describes the quality metrics used in the model for objective measurement of quality of the requirements document. Each quality demand for conformance criteria can have one or more quality metrics and some of these metrics can be common to two or more quality demands. The quality metrics for all the seven quality demands are described below [19, 20, 21, 22].

5.3.1 Completeness

1) **Metric Name** – Percentage of requirements labeled with TBD (To Be Determined).
   Input to Measurement – Requirements specification document.
   Assumption – TBD's are used in requirements specification document to indicate those requirements where necessary information is still required to be filled in to make them complete.
   Formula Used – $P_{TBD} = (N_{TBD} / N_{TOT})*100$ where N_{TBD} gives the number of requirements labeled with TBD and N_{TOT} gives the total number of requirements in the document.
   Boundary Value – 0 %. .

2) **Metric Nam**e – Percentage of requirements with no identifier.
   Input to Measurement – Requirements specification document.
   Assumption – Each requirement has a unique identifier (number) which can be used for referencing purposes and creating links between project-related documents.
   Formula Used – $P_{NUI} = (N_{NUI}/N_{TOT})*100$ where N_{NUI} gives the number of requirements

with no identifier.
Boundary Value – 0 %.

3) **Metric Name** – Percentage of requirements referring to objects which are not present.
Input to Measurement – Requirements specification document and other project-related documents.
Assumption – A requirement in the document can refer to one or more objects (e.g., requirements, tables, figures, sections, documents) in the same document or other project-related documents.
Formula Used – $P_{RI} = (N_{RI}/N_{TOT})*100$ where N_{RI} gives the number of requirements which refer to the objects (not present) in the same document or other project-related documents.
Boundary Value – 0 %.

4) **Metric Name** – Number of use-cases and actors with no names.
Input to Measurement – Requirements specification document with use-case diagrams and scenario descriptions.
Assumption – Each use-case and actor should have a unique name relevant to the project, which can be used to identify it in the use-case diagram and scenario description.
Formula Used – $N_{USN} + N_{ASN}$ where N_{USN} gives the number of use-cases without any name and N_{ASN} gives the number of actors without any name.
Boundary Value – 0.

5) **Metric Name** – Number of incomplete scenario descriptions.
Input to Measurement – Requirements specification document with use-case diagrams and scenario descriptions.
Assumption – A scenario description is an elaboration of a use-case into a precisely defined statement of system behavior. A scenario description is said to be incomplete if some of the necessary information is missing (e.g., pre-conditions, post-conditions).
Formula Used – N_{UCIS} where N_{UCIS} gives the number of incomplete scenario descriptions. **Boundary Value –** 0.

6) **Metric Name** – Number of unused use-cases and actors.
Input to Measurement – Requirements specification document with use-case diagrams and scenario descriptions.
Assumption – Unused use-case is defined as the use-case that is not associated with any actors, or included in or extending other use-cases. Similarly, an actor that is not associated with any use-cases is called unused actor.
Formula Used – $N_{UUC +} N_{UA}$ where N_{UUC} gives the number of unused use-cases and N_{UA} gives the number of unused actors.
Boundary Value – 0.

7) **Metric Name** – Percentage of requirements containing a number with no units of measurement attached.
   Input to Measurement – Requirements specification document.
   Assumption – Requirements may contain numbers. For example, requirements about the response time, recovery time of a system will contain numbers with units of time attached to them.
   Formula Used – $P_{NRU} = (N_{NRU}/N_{TOT})*100$ where N_{NRU} gives the number of requirements containing a number with no units of measurement attached.
   Boundary Value – 0 %.

8) **Metric Name –** Percentage of requirements containing one or more project-related terms/ abbreviations/acronyms which are not explained in the glossary.
   Input to Measurement – Requirements specification document with glossary.
   Assumption – Glossary is used to explain all the terms that are specific to the project or application domain. It also explains acronyms, abbreviations or other project or company specific shorthands.
   Formula Used – $P_{RAG} = (N_{RAG}/N_{TOT})*100$ where N_{RAG} gives the number of requirements with unexplained project-related terms/abbreviations/acronyms.
   Boundary Value – 0 %.

9) **Metric Name –** Percentage of requirements that do not have a test case.
   Input to Measurement – Requirements specification document and test plan document.
   Assumption – A traceability matrix (forward) can be compiled between software requirements and test cases, which help in identifying the requirements for which no test cases have been defined.
   Formula Used – $P_{NRTG} = (N_{NRTG}/N_{TOT})*100$ where N_{NRTG} gives the number of requirements with no test cases.
   Boundary Value – 0 %.

10) **Metric Name** – Percentage of requirements that are out of scope.
   Input to Measurement - Requirements specification document and business requirements document.
   Assumption – A traceability matrix (backward) can be compiled between software requirements and business requirements, which help in identifying the requirements that are out of scope. In other words these requirements do not map to any business requirements and are unnecessary features which should not be implemented.
   Formula Used – $P_{OSR} = (N_{OSR}/N_{TOT})*100$ where N_{OSR} gives the number of requirements that do not map to any business requirements.
   Boundary Value – 0 %.

Similarly traceability matrix can also be complied between business requirements and software requirements to identify the business objectives that are not mapped to software requirements (requirements for those objectives are missing).

5.3.2 Consistency

1) **Metric Name** - Percentage of redundant requirements.
   Input to Measurement – Requirements specification document.
   Assumption – A requirement may be present more than once in the document. This repetition may help in making the document more understandable, but increases the potential for injecting errors (e.g., redundant requirements may be overlooked and not updated). Such requirements need to be analyzed to see if they are really the same and if different they should be rewritten to make their differences clear. Cross-references should be used in place of repetition to avoid such problems.
   Formula Used – $P_{PRR} = (N_{PRR}/N_{TOT})*100$ where N_{PRR} gives the number of requirements which are already present in the document.
   Boundary Value – 0 %.

2) **Metric Name** – Percentage of requirements that cannot be uniquely identified.
   Input to Measurement - Requirements specification document.
   Assumption – Each requirement has a unique identifier that can be used to refer it in the document and also to create links between project-related documents.
   Formula Used – $P_{NUIR} = (N_{NUIR}/N_{TOT})*100$ where N_{NUIR} gives the number of requirements having an identifier already used by some other requirement in the document.
   Boundary Value – 0 %.

3) **Metric Name** – Number of use-cases referencing themselves.
   Input to Measurement - Requirements specification document with use-case diagrams and scenario descriptions.
   Assumption – A use-case may refer to itself directly or indirectly through inheritance and include/extend dependencies between it and other use-cases.
   Formula Used – N_{LUC} where it gives the number of use-cases which refer to themselves in the document.
   Boundary Value – 0.

4) **Metric Name** – Percentage of requirements referring to objects containing unexpected information.

Input to Measurement - Requirements specification document and other project-related documents.

Assumption – A requirement in the document may refer to one or more objects (e.g., elements/tables/figures/sections/documents) in the same document or other project-related documents. An object is said to be wrongly referred, if it does not provide the desired or expected information.

Formula Used – $P_{RROW} = (N_{RROW}/N_{TOT})*100$ where N_{RROW} gives the number of requirements referring to wrong objects.

Boundary Value - 0 %.

5) **Metric Name** – Percentage of requirements having terms/abbreviations/acronyms, which have the same explanation as that of some other term/abbreviation/acronym used in the requirements document.

   Input to Measurement – Requirements specification document with glossary and a dictionary of terms (project-related) along with their synonyms.

   Assumption – Different terms/acronyms/abbreviations may be used in the requirements document having the same explanation, which may inject defects in the product if they are misinterpreted by the developer.

   Formula Used – $P_{TAA} = (N_{TAA}/N_{TOT})*100$ where N_{TAA} gives the number of requirements which used different terms/abbreviations/acronyms with the same explanation.

   Boundary Value – 0 %.

6) **Metric Name** – Percentage of requirements having spelling errors.

   Input to Measurement – Requirements specification document.

   Assumption – Requirements may contain spelling errors, which need to be checked and corrected using a standard dictionary and a project-related dictionary of words.

   Formula Used – $P_{SPL} = (N_{SPL}/N_{TOT})*100$ where N_{SPL} gives the number of requirements with one or more spelling errors.

   Boundary Value – 0 %.

7) **Metric Name** – Percentage of requirements having grammatical errors.

   Input to Measurement – Requirements specification document.

   Assumption – Requirements may contain grammatical errors, which need to be checked and corrected with respect to language rules.

   Formula Used – $P_{GME} = (N_{GME}/N_{TOT})*100$ where N_{GME} gives the number of requirements with grammatical errors.

   Boundary Value – 0 %.

8) **Metric Name** – Percentage of requirements using words from more than one language.

Input to Measurement – Requirements specification document.
Assumption – In outsourcing projects, it is possible that the outsourcing party used some words or writes some portion of the requirements document in the native language. This can inject defects in the product, as the vendor party may misinterpret these requirements. Requirements document should be written in a language well understood by both the parties to avoid any kind of misinterpretations related to language.
Formula Used – $P_{DFL} = (N_{DFL}/N_{TOT})*100$ where N_{DFL} gives the number of requirements which include words from more than one language.
Boundary Value – 0 %.

Quality metrics have not been defined for consistency problems related to semantics (e.g., response time shall be 3 inches), but these defects can be identified by establishing a set of rules which determine whether the requirements can be labeled as semantically consistent or not. Similarly semantic contradictions in the requirements document are hard to detect and involve subjective interpretation. Experts assessing the quality of requirements can provide suggestions to write the requirements such that there is no place for semantic contradictions.

5.3.3 Unambiguity

1) **Metric Name** – Percentage of requirements which are written as optional sentences.
Input to Measurement – Requirements specification document and a list of optionality-revealing words (e.g., possibly, eventually, if needed, if appropriate).
Assumption – An optional sentence is one, which can or cannot be considered (e.g., the system shall be such that the mission can be pursued, possibly without performance degradation). Requirements should not be written as optional sentences as they increase the level of ambiguity of the document.
Formula Used – $P_{ONS} = (N_{ONS}/N_{TOT})*100$ where N_{ONS} gives the number of requirements that are written as optional sentences.
Boundary Value – 0 %.

2) **Metric Name** – Percentage of requirements which are written as subjective sentences.
Input to Measurement – Requirements specification document and a list of subjectivity-revealing words (e.g., similar, having in mind, better).
Assumption – A subjective sentence is one, which refers to personal opinions and feeling (e.g., the system shall be as far as possible composed of efficient software components).
Formula Used – $P_{SST} = (N_{SST}/N_{TOT})*100$ where N_{SST} gives the number of requirements that are written as subjective sentences.
Boundary Value – 0 %.

3) **Metric Name** – Percentage of requirements which are written as vague sentences.
   Input to Measurement – Requirements specification document and a list of vagueness-revealing words (e.g., bad, good, fast, recent, close).
   Assumption – A vague sentence is one, which includes one or more words having a non uniquely quantifiable meaning (e.g., the design shall be easy to modify).
   Formula Used – $P_{VG} = (N_{VG}/N_{TOT})*100$ where N_{VG} gives the number of requirements that are written as vague sentences.
   Boundary Value – 0 %.

4) **Metric Name** – Percentage of requirements which are written as weak sentences.
   Input to Measurement – Requirements specification document and a list of weakness-revealing verbs (e.g., can, could, may).
   Assumption – A weak sentence is one, which contains a weak main verb (e.g., the system may report the results in a separate output file).
   Formula Used – $P_{WKS} = (N_{WKS}/N_{TOT})*100$ where N_{WKS} gives the number of requirements that are written as weak sentences.
   Boundary Value – 0 %.
   There are no quality metrics defined for syntactic and semantic ambiguities as it is difficult to identify them by using a list of words which can make a sentence syntactically or semantically ambiguous. In some cases these ambiguities are easy to identify (e.g., the system shall not remove faults and restore service – syntactically ambiguous) but it is difficult to give a set of rules against which every requirement can be checked.
   Experts based on their experiences can provide suggestions to rewrite these requirements such that they do not have multiple interpretations.

5.3.4 Understandability

1) **Metric Name** – Percentage of non-atomic requirements.
   Input to Measurement – Requirements specification document and a list of conjunctive and disjunctive words.
   Assumption – Requirements having a too complex syntactical structure (requirements containing conjunctions, disjunctions) are hard to be understood. For example, “the battery low warning lamp shall light up when the voltage drops below 3.6 volts, and the current workspace or input data shall be saved”. From this sentence it is not clear what should be saved, and when. Some dangerous conjunctions/disjunctions include: and, or, also, with, either.
   Formula Used – $P_{NATC} = (N_{NATC}/N_{TOT})*100$ where N_{NATC} gives the number of non-atomic requirements in the document.

Boundary Value – 0 %.

2) **Metric Name** – Percentage of requirements which are written as implicit sentences.
Input to Measurement – Requirements specification document and a list of words that should not be used.
Assumption – An implicit sentence is one where the subject is generic rather than specific. The presence of implicit requirements makes the requirements document prone to be misunderstood. For example, these requirements shall be verified by using test cases. This sentence does not specify which requirements (also how many) should be verified using test cases.
Formula Used – $P_{IPL} = (N_{IPL}/N_{TOT})*100$ where N_{IPL} gives the number of requirements which are written as implicit sentences.
Boundary Value – 0 %.

3) **Metric Name** – Percentage of under-specified requirements.
Input to Measurement – Requirements specification document and a list of words that need to be more specified, and should not be used in the requirements document.
Assumption – A requirement is said to be under-specified if the subject of the requirement contains a word identifying a class of objects without a specifier of this class. For example, all managers shall be gives access to the product file. This sentence does not specify what kind of access (e.g., read, write, execute, remote) should be given to all the managers. These requirements can be misunderstood easily by the developer, and implemented wrongly in the product.
Formula Used – $P_{USPR} = (N_{USPR}/N_{TOT})*100$ where N_{USPR} gives the number of requirements that are underspecified.
Boundary Value – 0 %.

4) **Metric Name** – Flesch reading ease and Flesch-Kincaid grade level.
Input to Measurement – Requirements specification document.
Assumption - In the Flesch reading ease test, higher scores indicate material that is easier to read; lower numbers mark passages that are more difficult to read. Flesch–Kincaid grade level formula translates the 0-100 score to a U.S. grade level, making it easier ot judge the readability level.
Formula Used - Flesch reading ease score:

$$206.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right)$$

Flesch-Kincaid grade level:

$$0.39\left(\frac{\text{total words}}{\text{total sentences}}\right)+11.8\left(\frac{\text{total syllables}}{\text{total words}}\right)-15.59$$

Boundary Value – Above 50 for Flesch reading ease and below 12 for Flesch-Kincaid grade level.

For example to compute the Flesch-Kincaid grade level of the text give on a page of a requirements document following steps need to be taken:

- Count the total number of words and sentences in the text.
- Calculate the average number of words per sentence and multiply the result by 0.39.
- Count the total syllables, where a syllable is the smallest sound set people can make when they speak. A syllable has one and only one vowel. Most syllables have consonants too. If a syllable ends with a consonant, it is called a closed syllable. If a syllable ends with a vowel, it is called an open syllable. Calculate average number of syllables in words.
- Multiply the average syllable by 11.8
- Add the two results and subtract 15.59.

The result will be a number that equates to a grade level. For example, a 6.5 is a sixth grade reading level result.

5.3.5 Testability

1) **Metric Name** – Percentage of requirements that do not have a test case.
   Input to Measurement – Requirements specification document and test plan document.
   Assumption – A traceability matrix (forward) can be compiled between software requirements and test cases, which help in identifying the requirements for which no test cases have been defined.
   Formula Used – $P_{NRTG} = (N_{NRTG}/N_{TOT})*100$ where N_{NRTG} gives the number of requirements with no test cases.
   Boundary Value – 0 %.

2) **Metric Name** – Number of unnecessary test cases present in the test plan document.
Input to Measurement - Requirements specification document and test plan document.
Assumption – A traceability matrix (backward) can be compiled between test cases and software requirements, which help in identifying the test cases which are unnecessary (extra), and can be removed from the document. These test cases cannot be mapped to any of the requirements in the requirements document.
Formula Used – N_{UNTC} where N_{UNTC} gives the number of test cases in the test plan documents that cannot be mapped to any of the requirements.
Boundary Value – 0.

5.3.6 Modifiability

1) **Metric Name** - Percentage of redundant requirements.
Input to Measurement – Requirements specification document.
Assumption – A requirement may be present more than once in the document. This repetition may help in making the document more understandable, but it increases the potential for injecting errors (e.g., redundant requirements may be overlooked and not updated). Such requirements need to be analyzed to see if they are really the same and if different they should be rewritten to make their differences clear. Cross-references should be used in place of repetition to avoid such problems.
Formula Used – $P_{PRR} = (N_{PRR}/N_{TOT})*100$ where N_{PRR} gives the number of requirements which are already present in the document.
Boundary Value – 0 %.

Some other checks can be performed to make sure that the structure and style of the document is such that changes can be performed easily, completely and consistently. Related requirements should be grouped together (e.g., by stakeholder) so that the effects of changes made to one requirement can be reflected in others. Requirements document should also have a table of contents which becomes more important when the size of the requirements document is large.

5.3.7 Prioritization

1) **Metric Name** – Percentage of requirements that are not prioritized.
Input to Measurement – Requirements specification document.
Assumption – An implementation priority is assigned to each requirement, so that the most essential features are implemented first and low priority features are implemented only if sufficient time and resources are available.

Formula Used – $P_{PRTZ} - (N_{PRTZ}/N_{TOT})*100$ where N_{PRTZ} gives the number of requirements that are not assigned implementation priority.
Boundary Value – 0 %.

All the quality metrics defined have been assigned strict boundary values; this is due to the fact that these metrics are newly introduced, and need to be applied to projects from different IT domains to determine the more realistic boundary values. Some of these boundary values may be relaxed in future (e.g., what percentage of requirements labeled with TBD are allowed for the project to start) by analyzing the results of their application.

5.4 Assessment Process

Different steps involved in the assessment process are shown in the Figure 5-2 and explained below:

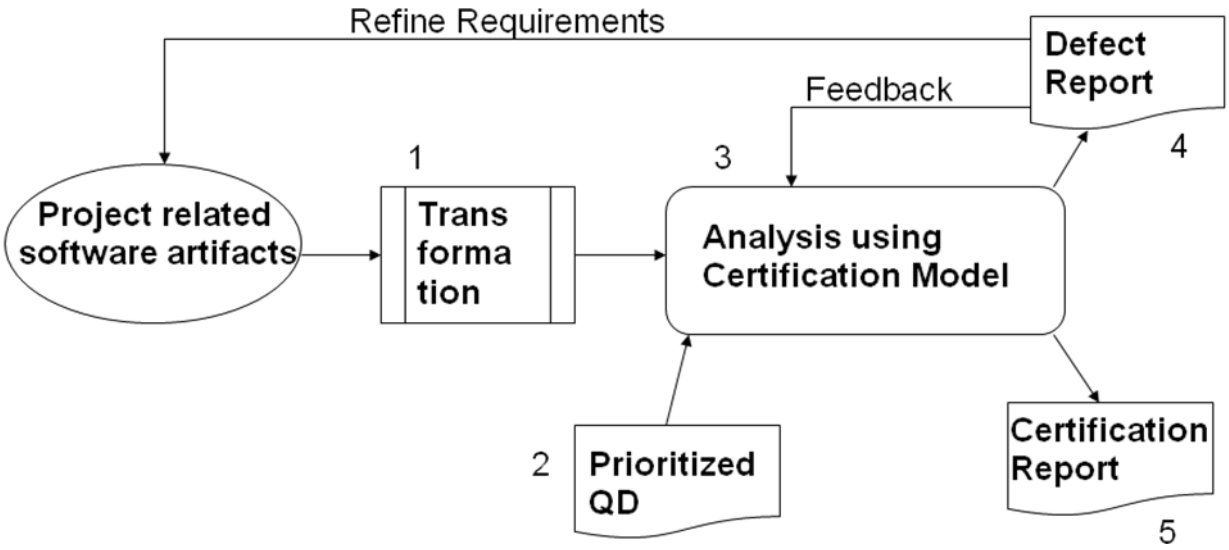

Figure 5-2 Steps of assessment process

1. **Transformation** – This step involves transforming the project related software artifacts (e.g., requirement specification document, test plan document, and business requirement document) to a suitable state that can be assessed by the model. Different activities performed in this step includes labeling requirements as functional, non-functional, constraint, and behavioral properties, compiling traceability matrices, filling out the

requirements and use-case templates, writing of requirements with commercial tools (depending on the project complexity) and generating the natural language requirements document in automatic way. The new state obtained should get approved by the customer to make sure none of the fields in the original document have been misinterpreted, as the certification level will be given based on this transformed state.

2. **Prioritized quality demand document** – as explained earlier based on the inputs from the customer or outsourcing party and the aspects of software development that are important for the project, elements/documents and quality demands (conformance) should be prioritized. Along with giving prioritization of quality demands and elements/documents, this document will also explain and justify the need for this particular prioritization.

3. **Analysis using certification model** – This step involves applying the proposed quality model to the project and measuring the quality objectively using quality metrics. Requirements quality can be measured automatically (along with manual measurement of some quality metrics) using tools such as ARM, QuARS, and DESIRe, which are explained the Chapter 6.

4. **Defect report** – if the requirements cannot achieve the desired certification level due to presence of defects/errors, a defect report would be generated. This report will describe in detail the different types of defects that are present in the requirements document and will also provide suggestions to improve the quality of requirements as shown in Figure 5-2. It also provides feedback to the certification model, to adjust the boundary values of quality metrics and add new quality checks if required.

5. **Certification report** – A desired certification level will be handed over if no defects are present in the requirements document. This report will describe the quality checks that have been performed on the requirements document. For each specific quality criteria, level achieved will be present in the report along with overall certification level. For conformance criteria, quality metric values achieved can also be included in the certification report.

5.5 New Certification Levels

Subsection 5.5.1 describes the new Certification Criteria Achievement Levels that apply for software requirements. The new overall certifications levels are explained in Subsection 5.5.2.

5.5.1 Criteria Specific Achievement Levels

For each of the three Certification Criteria new Achievement Levels can be established, as summarized in Table 5-2 (Availability), Table 5-3 (Uniformity), and Table 5-4 (Conformance).

CC1	**Availability**
0	Some high priority elements are missing
1	Some medium priority elements and/or documents are missing
2	Some low priority elements and/or documents are missing
3	All the elements and documents are present

Table 5-2 Achievement levels for availability

CC2	**Uniformity**
0	No standardization
1	Within the product
2	Style complies to a company standard
3	Style complies to an industry standard

Table 5-3 Achievement levels for uniformity

CC3	Conformance
0	Some high priority quality demands are not satisfied
1	Some medium priority quality demands are not satisfied
2	Some low priority quality demands are not satisfied
3	All quality demands are satisfied

Table 5-4 Achievement levels for conformance

5.5.2 Certification Levels

The model has eight certification levels. The first one is only use for creating a baseline that the certification analysis can start. The certification levels take into account the availability of project-related documents and are based on the prioritized quality demands unlike LSPCM where certification level is calculated based on the type of verification preformed on the requirements document. The maturity of requirements document is better reflected in the achievement levels and in the overall certification level as all the three quality criteria have their influence on the certification. These overall certification levels can be calculated from the levels achieved for the three certification criteria as follows:

1. CC1 = 1 and CC2 ≥ 1 and CC3 = 0

All the high priority elements are present, and the elements are uniform.

2. CC1 = 1 and CC2 ≥ 1 and CC3 = 1

All the high priority elements are present, uniform, and only high priority quality demands are satisfied.

3. CC1 = 2 and CC2 ≥ 1 and CC3 = 1

All the high and medium priority elements and documents are present, uniform, and only high priority quality demands are satisfied.

4. CC1 = 3 and CC2 ≥ 1 and CC3 = 1

All the elements and documents are present, uniform, and only high priority quality demands are satisfied.

5. CC1 = 1 and CC2 ≥ 1 and CC3 = 2

All high priority elements are present, uniform, and all the high and medium priority quality demands are satisfied.

6. CC1 = 2 and CC2 ≥ 1 and CC3 = 2

All high and medium priority elements and documents are present, uniform, and all the high and medium priority quality demands are satisfied.

7. CC1 = 3 and CC2 ≥ 1 and CC3 = 2

All elements and documents are present, uniform, and all the high and medium priority quality demands are satisfied.

8. CC1 = 2 and CC2 ≥ 2 and CC3 = 3

All high and medium priority elements and documents are present, comply with company standards, and all the quality demands are satisfied.

9. CC1 = 3 and CC2 ≥ 2 and CC3 = 3

All the elements and documents are present, comply with company standards, and all the quality demands are satisfied.

Certification levels give more weight to conformance quality criteria, as it determines if the requirements document satisfy the quality demands that are important for the project. Certification level increases with the increase in the number of elements/documents satisfying the high priority quality demands. Level 8 does not start with CC1 = 1, as it is not realistic that all the quality demands are satisfied when only high priority elements are present in the requirements document (e.g., if test plan document is not present, testability cannot be measured). In the last two levels, CC2 (Uniformity) should be at least 2 to make sure that highest certification levels are given only when requirements are written according to company or industry standards.

6. Requirements Measurement Tools

This chapter describes some tools that can be used to measure the quality of requirements for a project. Requirements quality metrics can be measured using these tools as they help in reducing the time required for certification. Though complete automation of quality measurement process has not been achieved in the proposed model, but these tools can be configured and applied to most of the quality metrics.

6.1 Automated Requirements Measurement Tools (ARM)

The objective of the Automated Requirements Measurement Tools (ARM) is to provide measures that can be used to assess the quality of a requirements specification document [23]. ARM is not intended to be used for the evaluation of the correctness of a specified requirements document. This tool can be seen, as an aid for “writing the requirements right”, not “writing the right requirements”.

It was developed by the Software Assurance Technology Center (SATC) at the NASA Goddard Space Flight Center as an early life cycle tool for assessing requirements that are specified in natural language.

The quality model of ARM was defined by compiling first a list of quality attributes that requirements specifications are expected to exhibit, then a list of those aspects of a requirements specification that can be objectively and quantitatively measured. The two lists were analyzed to identify relationships between what can be measured and the desired quality attributes. This analysis resulted in the identification of categories of sentences and individual items (i.e., words and phrases) that are primitive indicators of the specification’s quality and that can be detected and counted by using the document text file. These indicators have been grouped according to their indicative characteristics.

Using these indicators, the ARM tool creates a file that includes three reports. This file contains a Summary Report, a detailed Imperative Report and a detailed Weak Phrase Report. The ARM Summary Report includes the total number of times each quality indicator occurs in the requirements document. The location within the source file of each specification statement identified by the tool and a copy of the specification statement are listed by the Imperative Report. The Weak Phrases Report lists the location and specifications that contain indicators that are considered to be phrases that weaken the specification.

The categories related to individual specification statements are:

- Imperatives

- Directives
- Weak Phrases
- Continuances
- Options
- Incompletes

The categories of indicators related to the entire requirements document are:

- Size
- Readability
- Specification Depth
- Text Structure

This tool can be applied to assess the quality metrics described in Chapter 5. For example, categories related to individual specification statements such as options and weak phrases can be used to assess metrics 1 and 2, 3, 4 described in Section 5.3.3. The weak phrases list of ARM can be extended to include subjectivity, vagueness and weakness revealing words which are explained in Section 5.3.3. Similarly incompletes (e.g., TBD, TBE, and TBC) can be used to assess metric 1 described in Section 5.3.1.

6.2 Dynamic Expert System for Improving Requirements (DESIRe)

It is an expert system which helps in improving the quality of requirements for a project, independently and with help of a tool [24]. DESIRe is based on a knowledge database, which can be adjusted and expanded by developers according to the area of application. In this way, DESIRe creates an improved knowledge flow between developers regarding improving the natural language requirements.

It is a method to automatically identify words within the description text of a requirement and to indicate predefined questions, remarks and information in a window according to those previously identified words. These questions, remarks and information indicate possible weaknesses of the requirement. The author of the requirement will then try to answer the questions asked and if need be, he can rework the contents of his requirement. Thus he can assure that the requirement rules concerning completeness, unambiguousness and understanding are observed.

The questions, remarks and information are defined in a word/verb list, which may also be changed by the author. This word/verb list is based on following categories of words:

- *Indicational words* - are potentially weak words in textual requirements. These could be filler words (such as: also, thus ...and so on) or weak words (such as: but, all, anyone, at the most...and so on).

- *Verbs* - Should a verb be included in a requirement, questions may be asked based on this verb, questions which could possibly indicate gaps within the requirement.

- *Project-specific words* - The requirement authors can identify by themselves the nouns within their terminology and enter the related questions and remarks into the list of questions, represented by a list of words.

- *Consistent use of terms* – it can support the consistent use of terms. For this the DESIRe word list is used as glossary, where the meaning of the terms is uniquely defined. Whenever DESIRe finds a term defined in the glossary within a requirement, this definition from the glossary is shown and the user can verify if he has used the term according to definition.

DESIRe can be used to assess the quality metrics by adjusting and expanding the knowledge database on which it is based. For example metrics 1, 2, 3, and 4 described in Section 5.3.3 can be assessed by expanding the *indicational words* list. Similarly metric 3 of Section 5.3.4, and metric 5 described in Section 5.3.2 can be assessed by expanding the *verbs* list and *consistent use of terms* list respectively.

6.3 Quality Analyzer for Requirements Specifications (QuARS)

The quality model of QuARS is composed of a set of high-level quality properties for NL requirements to be evaluated by means of syntactic and structural indicators [20, 25, 26]. The indicators are collected into specific dictionaries that contain terms and linguistic constructions characterizing a particular defect and directly detectable looking at the sentences of a requirements document. The quality model has been defined by considering existing related literature and by taking advantage from matured experience in the field of requirement engineering.

This quality model, though not exhaustive, is sufficiently specific to include a significant part of lexical and syntax-related issues of requirements document.

QuARS has been designed with the aim to match mainly the following characteristics:

- Ease to use – the tool has to be usable with little effort both in terms of people training needed and time consumed.
- Generality – the tool shall run independently of the format of the SRS document. For this reason the expected format of the requirement document to be evaluated is the text format. This format allows achieving a high generality because it is always possible to save every electronic format as text format.
- Tailorability/flexibility – the tool is modifiable in order to make it more effective for particular application domains. It could be necessary to adapt QuARS to different projects and different application domains. It allows to evolve and modify the dictionaries.
- Multi-lingual – the modularity of the structure of the tool allows to analyze requirements documents written in different languages simply by changing the lower level components of the tool and by translating the dictionaries in the new target language.

This tool can be applied to various quality metrics described in Chapter 5. Some of the quality metrics which can be directly assessed using QuARS are metric 8 of Section 5.3.1, metrics 1, 2, 3 and 4 described in Section 5.3.3. Similarly metrics 1, 2 and 3 of Section 5.3.4 can be directly measured using this tool. Table 6-1 describes the various tools that can be used to assess the quality metrics described in Chapter 5.

Quality Metrics	**Tools Used**
Metric 1 for Completeness	ARM, DESIRe
Metric 8 for Completeness	DESIRe
Metric 5 for Consistency	DESIRe
Metrics 6, 7 for Consistency	Word Processor
Metric 8 for Consistency	DESIRe
Metrics 1, 2, 3 and 4 for Unambiguity	ARM, DESIRe, QuARS
Metrics 1, 2 and 3 for Understandability	QuARS, DESIRe
Metric 4 for Understandability	Flesh 2.0 for Windows

Table 6-1 List of tools used for assessing quality metrics

7. Validation : Horus IMSETY Case Study

7.1 Introduction

Interface Making Satellite Experiment Tracking easy (IMSETY) will provide scientists and observers with an easy to use interface to conduct experiments on space based payloads and possible reference payloads on earth [27, 28].

The system will provide the means to compose, schedule and observe experiments and intercede with these schedules in real time. The system will communicate to the users whether or not this scheduling and these intercessions are possible at any given moment in time. The project was carried out by a team of software engineering students at Eindhoven University of Technology, Eindhoven in 2007.

This chapter briefly describes the defects that were found in the user requirement document for this project based on the application of proposed model to it. In the end some suggestions have been provided to improve the quality of requirements for this project.

7.2 Summary of Evaluation

This section describes the defects that were present in the user requirement document for the Horus IMSETY project. In order to assess the quality of requirements some assumptions have been made regarding the prioritization of quality demands which are as follows:

- Prioritization, unambiguity and consistency have been assigned the highest priority.
- Testability and completeness have been assigned the medium priority.
- Modifiability and understandability have been assigned the lowest priority.

Defects present in the user requirement document are as follows:

1) Quality criteria – Availability.
   Problem – Business requirement document is missing.
   Description – Business requirement document has not been provided along with user requirement document and acceptance test plan document and therefore traceability matrix cannot be compiled between business requirements and user requirements.

2) Quality criteria – Conformance.

Problem – Completeness.
Description – Use-case diagrams are not provided with any of the use-case descriptions.

3) Quality Criteria – Conformance.
   Problem – Completeness.
   Description – Exception flow field is missing in all the use-case scenario descriptions making the flow of events which will happen in the case of the error unclear.

4) Quality Criteria – Conformance.
   Problem – Completeness.
   Description – Post conditions field is missing in all the use-case scenario descriptions making what the change in state of the system will be after the use-case completes unclear.

5) Quality Criteria – Conformance.
   Problem – Readability.
   Description **–** Flesh reading ease score is 45.46 for complete document from Section 1 which is below the boundary value.

7.3 Suggestions for Improving the Quality of Requirements

A few suggestions to improve the quality of requirements for the Horus IMSETY project are as follows:

- Use-case diagrams (drawn according to UML standards) should be provided along with detailed scenario description for each use-case.
- With every use-case scenario description, identifier of one or more requirements that are explained using that use-case should be provided.
- Requirements should be labeled as functional, non-functional, constraint requirements etc.
- Use-case scenario description should include post conditions and exception flow fields.
- Business requirements should be provided along with user requirements to identify the extra and missing requirements.

- Along with every user requirement information should be provided about the business requirement to which it belongs.

8. Conclusions and Recommendations

Requirements are the foundation upon which the entire system is built. The high failure rate of software indicates that often these requirements are not satisfied. This problem becomes more significant when the software projects are outsourced. This thesis proposes a flexible model which can provide the core for a new version of LSPCM which can be used for certifying software requirements of outsourcing projects. The new model provides answers to the research questions mentioned in the Section 1.1. The new version of LSPCM based on the proposed model includes quality checks imposed by those aspects which are important for outsourcing and will be able to objectively measure the quality of software requirements using quality metrics. It can also be adjusted to certify requirements for a particular IT/Business domain based on the prioritization of requirement elements, project-related documents and quality demands. Though quality metrics help in improving the quality of software requirements but manual measurement is prone to error and time consuming and therefore automated measurement has been used. The overall certification level is provided taking into account the availability of project-related documents and thus the maturity of software artifacts is better reflected. In order to validate the model it has been applied to a case study and the results of the validation prove that the model is effective in improving the quality of software requirements.

A few recommendations for further work are as follows:

- The proposed model needs to be applied to several outsourcing projects from different IT domains, as it would help in identifying some more important quality checks and quality metrics which should be included in the new version.
- Based on the analysis of these outsourcing projects, relationship between IT domains and prioritization of requirement elements, project-related documents and quality demands need to be made more concrete.
- The boundary values defined for quality metrics can be relaxed and more realistic values need to be included based on the analysis of various outsourcing projects from different IT domains.
- This thesis characterizes five IT domains, which can be further extended and the relationship between new IT domains and aspects should be established.
- More requirement measurement tools need to be explored, evaluated and applied, with the aim to achieve complete automation of quality metrics.

References

[1] J. M. Bhat, M. Gupta, and S. N. Murthy, "Overcoming requirements engineering challenges: Lessons from offshore outsourcing," *Software, IEEE*, vol. 23, no. 5, pp. 38-44, 2006. [Online]. Available: http://dx.doi.org/10.1109/MS.2006.137.

[2] J. Hanisch and B.J. Corbitt, "Requirements Engineering during Global Software Development: Some Impediments to the Requirements Engineering Process: A Case Study," *Proc. 12th European Conf. Information Systems* (ECIS 04), 2004. [Online]. Available: http://is2.lse.ac.uk/asp/aspecis/20040057.pdf.

[3] P. Heck, M. Klabbers, and M. van Eekelen, "A software product certification model," *Software Quality Journal*, Springer Netherlands, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11219-009-9080-0.

[4] "IEEE standard glossary of software engineering terminology," *Tech. Rep.*, 1990. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=159342.

[5] G. Kotonya, and I. Sommerville, *Requirements Engineering – Processes and Techniques*, John Wiley and Sons Ltd., 1998.

[6] E. Dubois, J. Hagelstein, and A. Rifaut, "Formal Requirements Engineering with ERAE," *Philips Journal Research*, Vol. & 43, No 4, 1989.

[7] R.F. Goldsmith, *Discovering Real Business Requirements for Software Project Success*, Artech House, 2004.

[8] I. F. Alexander, and R. Stevens, *Writing Better Requirements*, Addison-Wesley, 2002.

[9] K. E. Wiegers, *Software Requirements*, Microsoft Press, 2003.

[10] Why, What, Where and How of Outsourcing: White Paper, S5 Systems. [Online] Available: http://www.s5systems.com/whitepapers/whywhatwhereandhowofoutsourcing.pdf.

[11] E. Carmel, and P. Tija, *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*, Cambridge University Press, 2007.

[12] M. J. Mahemoff, and L. J. Johnston, "Software Internationalization: Implications for Requirements Engineering," *Proc. 3rd Australian Workshop on Requirements Engineering*, pp. 83-90, 1998.

[13] V. Imsland, and S. Sahay, "Negotiating Knowledge: The Case of a Russian-Norwegian Software Outsourcing Project," *Scandinavian J. Inf. Systems*, vol. 17, 2005. [Online]. Available: http://www.e-sjis.org/journal/volumes/volume17/articles/06imslandsahay.pdf.

[14] C. Jones, *Applied Software Measurement: Global Analysis of Productivity and Quality*, 3rd ed., McGraw-Hill Professional, 2008.

[15] S. W. Ambler, and L. L. Constantine, *The Unified Process Inception Phase: Best Practices in Implementing the UP*, Focal Press, pp. 206-215, 2000.

[16] P. Heck, and M. van Eekelen, *The LaQuSo Software Product Certification Model: (LSPCM)*, Technische Universiteit Eindhoven, 2008.

[17] E. Knauss, and C. E. Boustani, "Assessing the Quality of Software Requirements Specification,"*Proc. 16th IEEE International Requirements Engineering Conf.*, IEEE Computer Society, 2008.

[18] R. J. Costello, and D. Liu, "Metrics for Requirements Engineering," Journal of Systems and Software, 1995.

[19] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "Quality Evaluation of Software Requirements Specifications," *Proc. Software and Internet Quality Week Conf.*, 2000.

[20] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "An Automatic Quality Evaluation for Natural Language Requirements," *Proc. 7th International Workshop on RE: Foundation for Software Quality*, 2001.

[21] J. Allen, *Natural Language Understanding*, Benjamin/Cummings Pub. Co., 1987.

[22] A. Fantechi, S. Gnesi, G. Lami, and A. Maccar, "Application of Linguistic Techniques for Use Case Analysis," *Requirements Engineering Journal*, Vol. 8, Issue 3, Springer-Verlag, August 2003.

[23] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated Analysis of Requirement Specifications," *Proc. 19th Int. Conf. Software Engineering* (ICSE-97), Boston, MA, May 1997.

[24] HOOD DESIRe - http://www.hood-group.com/en/products/desire/

[25] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "The Linguistic Approach to the Natural Language Requirements Quality: Benefits of the use of an Automatic Tool," *Proc. 26th Annual NASA Goddard Software Engineering Workshop*, IEEE Computer Society, Greenbelt, Maryland, 2001.

[26] A. Bucchiarone, S. Gnesi, and P. Pierini, "Quality Analysis of NL Requirements: An Industrial Case Study," *Proc. 13th IEEE Int. Conf. on Requirements Engineering*, IEEE Computer Society, Washington, DC, 2005.

[27] Horus URD - http://wwwis.win.tue.nl/2R690/projects/horus/urd.pdf.

[28] Horus ATP - http://wwwis.win.tue.nl/2R690/projects/horus/atp.pdf.

Appendix A

Practical Assessment of Business and IT Requirements for Offshoring (Section 4.1)

Reflection on LSPCM

The current version of LSPCM was not able to identify a number of the issues in the requirement documents of offshored projects. Following are the issues which will lead to improvements in the LSPCM model:

1. In practice business requirements are gathered first which is then transformed into IT requirements. Therefore it is important to assess the quality of business and IT requirements before offshoring the development. In LSPCM the business context and IT requirements of a project is covered under the product areas "*Context description*", "*User requirements*" and "*High level design*". The certification types for the product areas "*Context description*" and "*High-level design*" is missing in the LSPCM document. Hence the complete assessment of business and IT requirements cannot be done with the current version of LSPCM.

2. A company "X" makes all its project documents complying with its own standards. Now company "X" wishes to offshore its development to a company "Y" which has its own standards. This means that the two companies may have different standards which might not have a common ground over which both can comply. Hence in the case of offshoring it is essential to check whether the prepared documents comply with an industry standard. But in LSPCM the compliance with industry standards under the ***uniformity checks*** is never considered for deciding the certification levels.

3. From the second case study we understood that the client was already convinced with having SAP as the solution for their problem. This means that the client is aware of the different the quality assured by SAP which led them to choose SAP as their solution. Then documentation of non-functional requirements at the early stage of requirements elicitation may not be essential. The non-functional requirements if any can be explored at a later phase of project

development. But in LSPCM we cannot proceed with the assessment unless the non functional requirement documents are present.

4. In LSPCM there are only checks on the consistency of use cases and their corresponding diagrams. For package implementation projects there are no use cases and instead there are processes and process diagrams whose consistencies must be checked.

5. If the diagrams in the requirements document do not follow any industry standard then there is no common understanding for the symbols used in these diagrams. There is a possibility for wrong interpretation when the documents are sent to the offshored end. LSPCM checks that the requirement documents are compliant with industry standards. But result of the check "compliance with industry standards" never considered for deciding the achievement level.

6. In the first case study we observed that there were inconsistencies in the use of language for specifying the business and project related terms because the GUI was expected to be build in Dutch whereas the offshored team language is English. For example in the GUI illustration it is "*Stagiair scherm*" and in its description it is "Intern screen". This was an example of a mix in the languages Dutch and English. The similarity is not obvious unless the developers are proficient in both the languages. LSPCM does not provide any checks for identifying the same.

7. During the SEMBA/IRMA assessment of the case studies we identified incomplete non-functional requirements. These incomplete parts were not identified by LSPCM.

8. During the SEMBA/IRMA assessment of the case studies we identified that the non-functional requirements were not prioritized. LSPCM does not have any check for checking whether the non-functional or functional requirements were prioritized.

9. From the SEMBA/IRMA assessment of the case studies we identified that the "*Client objectives and expectations*" were not found. This was important for offshoring since the offshored team does not have a direct interaction with the client and will be aware of the client expectations only through the documentations. LSPCM does not have a check in the **required elements** of *Context description* product area, which will check the availability of

such a "*Project initiation document*" which will document the client's vision, objective and expectations.

Following are the reflections on LSPCM model in order to assess the requirement documents of offshore projects effectively:

1. The certificate types for the product area "Context description" and "High-level design" should be included in the model. For example: For each certification criteria in LSPCM there are number of specific criteria. LSPCM does not specify which specific criteria should be satisfied in order to achieve any certification level for "Context description" and "High-level design" product area. In this research we focus on the business and IT requirement and these requirements are mapped in "Context description", "User requirement" and "High-level design" product are of LSPCM. The certification types of the "User requirement" product area are readily available. In similar way LSPCM should include the certification types for the product area "Context description" and "High-level design".
2. The compliance with industry standards under the uniformity checks of LSPCM is necessary to be considered for deciding the certification levels.
3. There should be an exception in the required elements under the specific criteria "Formalness" of the user requirements assessment of LSPCM, when dealing with a package implementation project. The non-functional requirements should be **not applicable** in the case of package implementation projects.
4. Whenever an assessment for a package implementation project is to be carried out the checks for use cases in LSPCM should change into checks for lowest level of processes which is atomic and directly conforms to a functional requirement.
5. In the specific criteria "Formalness" of any product area, the required elements should also include description of all the symbols used in the pictorial representation of process models or data models or flowcharts. This check should be mandatory if industrial standards are not used for making these models.
6. In the specific criteria "Formalness" of high level design product, the translated-terms dictionary should be a required element. The translated-term dictionary should contain all the business and project environment related terms from other language. For example: if

requirements document is written in English and it contains a number of business or project terms in Dutch and offshore team do not understand Dutch. In this case all the Dutch terms with its translation should be present in translated-terms dictionary.

7. In the specific criteria “Conformance” of high level design product a check should be added to check that all other language terms are present in the translated-terms dictionary.
8. The certification criteria “Formalness” of user requirement product area require presence of non-functional requirements. The non-functional requirement should be more detailed (a list of quality attribute for specific domain[1] can be added) and more checks for should be included in the “Conformance” specific criteria to check the consistency of non-functional requirement.
9. Check for prioritization of non-functional requirements should be added in the “Conformance” specific criteria of user requirements product area.
10. There should be a check in the required elements for client vision, objectives and expectations under the specific criteria “Formalness” of the context description product area of LSPCM.
11. LSPCM has a check for all the product area “The certification property is relevant and feasible”. The definition of the certification property should be elaborated more with example in order to improve the usability of the LSPCM model.
12. The check for compliance with industry standard in LSPCM should not be applicable if it is a package implementation project.

[1] Domain could be real-time software or information system or scientific software or internet application software.