Software Quality Problems in Requirement Engineering and Proposed Solutions for an Organization in Mauritius

Kanishka Gopal
Dept. of Computer Science and Engineering
University of Mauritius

Sneha Jadoo
Dept. of Computer Science and Engineering
University of Mauritius

Jyotsna L.P. Ramgoolam
Dept. of Computer Science and Engineering
University of Mauritius

Vimla Devi Ramdoo
Lecturer, Dept. of IT
Charles Telfair Institute
Mauritius

ABSTRACT
Requirement engineering is traditionally the first step carried out in any software project, precisely requirement elicitation followed by the requirements specification documentation. It is the requirements that principally dictate how the software should be designed and implemented. Consequently, failing to capture the right requirements in a clear and unambiguous manner become a challenge in the field of software development. The impact is directly felt in the quality of the software produced. This paper analyzes the software quality problems related with requirement engineering and the associated challenges with respect to a software company situated in Mauritius. In order to alleviate the problems, solutions have been proposed to overcome the difficulties encountered and hence enhance software quality.

General Terms
Requirement Engineering, Software Quality.

Keywords
Requirement Engineering, Software Quality, Requirement Engineers, Software Requirements, Requirement Inspection.

1. INTRODUCTION
The world today is governed by the use of software. Software has become pervasive in our commerce, culture and daily activities. From the simple act of generating a chart to conducting financial transactions online, the importance of software is felt by virtually everyone and in every field. Organizations today rely heavily on software for their daily functioning.

Business domains vary from company to company. As a consequence, different types of software are necessary to suit demands depending on the nature of business of the company. Therefore, the first step towards building software is to understand what the software needs to do, that is, the requirements of the software. For example, a point of sales (POS) system for a shopping store needs to allow the user to keep track of sales and also enable transactions to be effected in an efficient way. As a result, requirements form the baseline of any software project as the success of a software system depends on how well it fits the needs of its users and stakeholders [1] [2]. The process through which these requirements are determined is known as Requirement Engineering.

Many companies today focus a lot on requirement engineering since it is one of the factors contributing towards a good quality software [3] [4]. According to Computer Finance Magazine, errors in software requirements and software design documents are more frequent than errors in the source code itself. Moreover, defects introduced during the requirements and design phase are not only more probable but also are more severe and more difficult to remove [5]. Hence, it becomes crucial to have a proper and adequate requirement engineering process in place in an organization to collect, analyze and specify requirements correctly in order to ensure that the software meets the customer needs, and at the same time enhance the quality of the software.

2. SOFTWARE QUALITY – WHY IS IT SO IMPORTANT?
According to IEEE Standard 1633-2008, the quality of software can be defined as:

(A) The totality of features and characteristics of a software product that bear on its ability to satisfy given needs, such as conforming to specifications.

(B) The degree to which software possesses a desired combination of attributes.

(C) The degree to which a customer or user perceives that software meets his or her composite expectations.

(D) The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer.

By definition, software essentially satisfies the needs of the customer. If the end-product does not meet the requirements of the customer, even if it functions properly, it is of no use to the customer. From the customer's perspective, the software will not be a quality product if it does not meet its expectations. High quality software not only means customer satisfaction, but it also entails lower maintenance costs and an increase in the software's sales values. The software organization also takes advantage of a good reputation provided by the satisfaction of its customers. Consequently, the latter will push the quality standards even higher. As a result, software quality is the focus of any software project, from the viewpoint of the software engineers as well as the customers.

The Cost of Quality is a measure that quantifies the cost of control/conformance and the cost of failure of control/non-conformance. In other words, it sums up the costs related to prevention and detection of defects and the costs due to occurrences of defects [6]. The relative cost of fixing an error earlier in the software development life cycle is less costly and the cost increases significantly as progress is made along the life cycle [7] [8] [9] as shown by Figure 1.

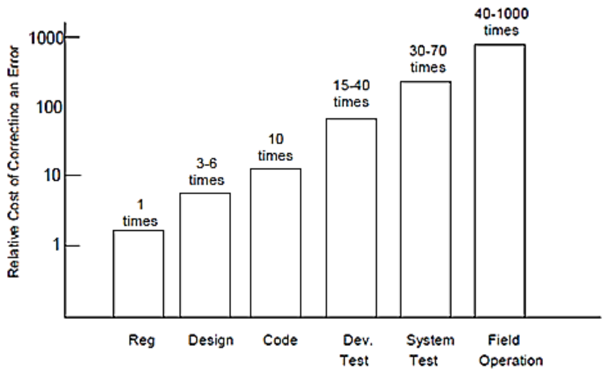

Fig 1: Relative Cost of Correcting an Error [9]

Thus, for any software development organization, software quality has an economic impact and this is the reason why quality is a leading concern. Ultimately, it will also cost less to the organization to build a quality software right from the initial phases of the software development life cycle.

Factors leading to poor quality in software can be controlled to a certain extent if appropriate considerations are given to the phases of the software development life cycle. Software quality can be predicted and controlled, but for that, it is essential that its causes are understood and addressed. For the context of this paper, the causes leading to poor software quality will be identified considering a particular software organization in Mauritius. Eventually, the cause having the largest stake in the quality of the software will be ascertained and solutions will be proposed in order to alleviate the problems faced by the software organization concerned.

3. METHODOLOGY

3.1 Company Profile

The company chosen for the purpose of this research is a state owned company, which designs and implements IT applications for the Government of Mauritius. The software development life cycle used is the traditional waterfall model. The main development platforms used by the company are Java and Oracle. Currently, at the start of any software project, requirements gathering are done by the developers of the software themselves. The team leader briefs the team of developers who conduct meeting sessions with the in order to collect the requirements. They are also the one responsible to prepare the Software Requirements Specifications (SRS) document for the software project. Often, developed software is of poor quality at the end of the software development life cycle, and is thus faced with many difficulties before actually going live.

3.2 Causes of Poor Software Quality

The main causes contributing to poor software quality have been identified using the Ishikawa or Fish-Bone diagram as depicted in Figure 2 and are elaborated as follows:

3.2.1 Unrealistic Schedule

The schedule for the project is calculated by the project manager and the latter rarely asks for the collaboration and input of the development team. The schedule rarely reflect reality as the majority of tasks take longer than the estimated time due to varying level of skills of the developers, and thus upsets the overall project’s schedule caused by the Domino Effect. Following schedule pressure, developers work under stress and are hence more prone to make errors. Thus, wrongly estimated schedules indirectly affect the quality of the software produced. This is also because slippage in schedules has an impact on the cost and scope of the project according to the traditional project management triangle.

3.2.2 Requirement Problem

Since requirements are gathered by the developers, they lack the required skills to negotiate and communicate with the client. Developers are not experts in the business domain of the clients and sometimes fail to understand the functional requirements of the system. The busy schedule and workload of the developers sometimes do not allow them to meet the clients to clear all misunderstandings. As a matter of fact, requirements inevitably change, even after the developers have started coding. Significant time and resources go into incorporating these changes by reworking on the codes. Continuous changes in scope make it difficult to balance the triple constraint of cost, time and scope. As a matter of fact, software quality is highly compromised and reduced [24].

3.2.3 Testing

Testing is conducted at nearly the last stages of the software development life cycle and by that time, most of the schedule that was planned is over. Lack of time prevents the testers to properly conduct testing. Absenteeism in the team also handicaps the testing process making it difficult to achieve good test coverage. At times, re-testing of the software once a bug has been corrected is not done properly due to schedule pressure.

3.2.4 People

The company comprises of a mix of experienced as well as inexperienced developers. To be able to integrate the team properly, new recruits need proper training which is given by the experienced developers themselves. Unfortunately, the developers are most of the time busy working on projects and thus unable to train the new recruits properly.

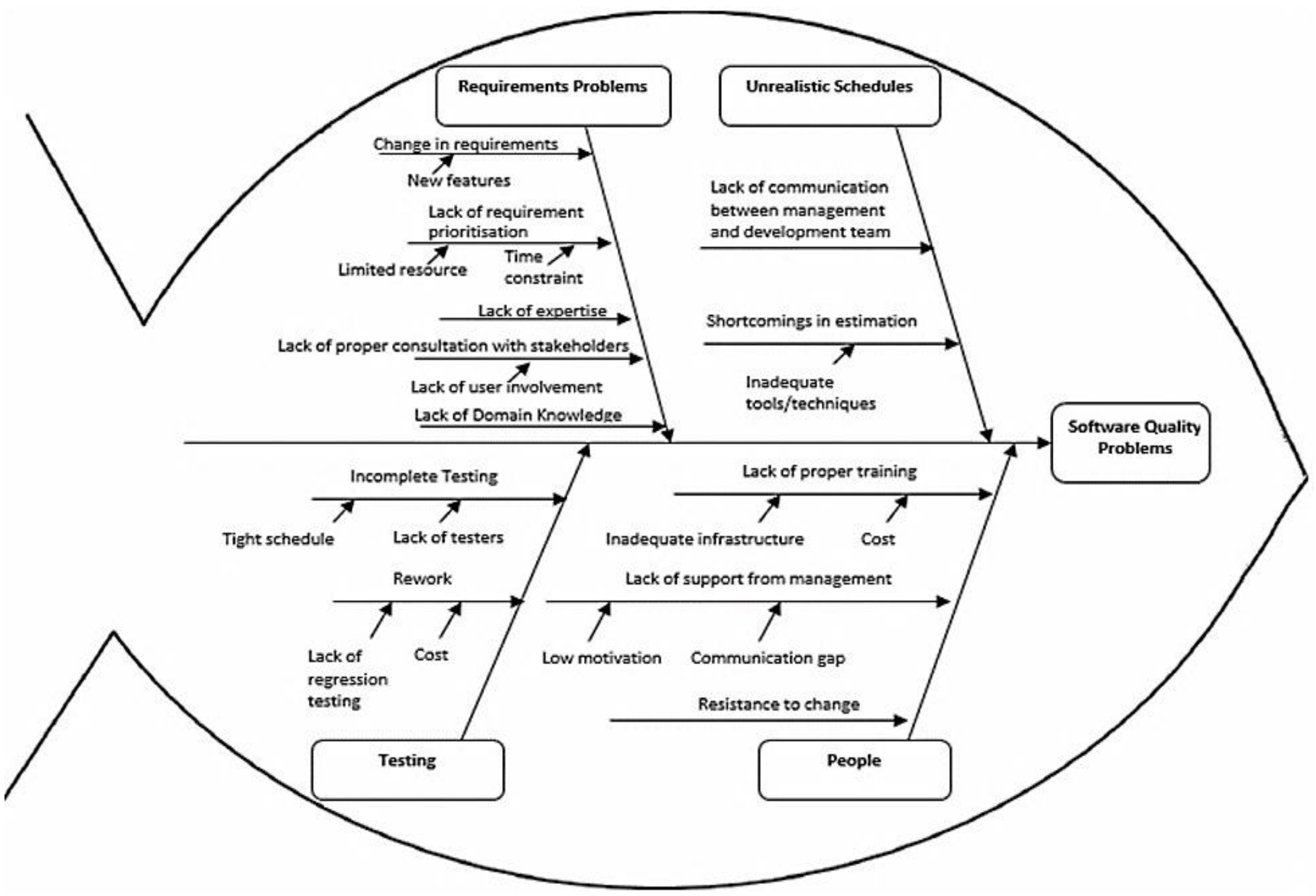

Fig 2: Potential causes of poor software quality (shown using the Ishikawa or Fish-Bone diagram)

3.3 Principal Cause of Software Quality Problems

To be able to improve the software quality in an efficient manner, it is imperative to know where to focus maximum efforts in order to solve the root cause of the problem.

The Pareto chart is one of the seven basic tools of quality control which can be used to quickly identify the major cause of poor software quality in projects. Accordingly, the Pareto chart has been used to come up with the major cause of poor quality software in the company concerned.

For the Pareto analysis, the following steps have been conducted:

1. A list of issues encountered in projects has been compiled using inputs from members of the company as shown in Table 1.
    - The company uses an online tool to log all bugs and issues that have been encountered for each project under a specific project ID. The issues consist of all change requests, rework requests, bug fixing requests and assistance requests from clients. A sample of two projects which deal with human resource management was chosen and the list of issues logged for those projects has been analyzed. After the analysis, similar issues have been grouped under a specific problem.
2. For each of the problems, a root cause (one of the main causes identified in the Ishikawa/Fish-Bone diagram in Figure 2) has been assigned as shown in Table 2.
3. A score has been assigned to each problem based on the frequency the problem occurred as shown in Table 3.
    - The aim of this research is to improve the quality of the software which will ultimately increase customer satisfaction. Therefore, the method used for scoring each problem is based on the number of times the problem occurred, that is, the number of times the issue or complaint has been logged on the online tool. Lowering the number of complaints and issues will help enhance software quality.
4. The scores for each root cause are added and the Pareto chart is plotted as depicted in Figure 3.

Table 1. Analysis of logged bugs (Step 1)

No	**Problems**	**Analysis of Issues**	**No. Of Logged Issues**
1	Client logged request since a long time and request was serviced only later.	Client requested for assistance with some functions. Enhancement to certain functions was also requested. Due to lack of resources (developers), the requests were attended at a later date.	5
2	Staff lacking experience was assigned for bug fixing.	Inexperienced developers were assigned for bug fixing. Thus, developers took a lot of time to first understand the bug and then correct it. Intervention from senior developers was also often needed on several issues.	4
3	Absence of some requirements in the SRS document.	Some requirements were not specified in the beginning. For instance, requirements for a lot of reports had to be specified in later stages of the software development life cycle.	7
4	Incomplete requirements.	This resulted in several errors in coding the requirements. For example, the difficulty in assignment of grades, salary scales to grades and refund of leaves for specific grades due to missing information from client.	3
5	Misinterpretation of requirements by developers.	Developers implemented some functionalities that did not match the requirements of the client. Requirements were developed from the developer’s perspective and understanding, leading to misinterpretation which came to the fore only after development.	6
6	Misunderstanding between client and developers.	Failure to take into consideration the clients’ perspective due to limited communication caused developers to have some misunderstanding about the requirements, resulting in re-designing of some parts of the system.	3
7	Changes in requirements after coding started.	Once the client was acquainted with part of the system, some requirements were changed since it was not what they really wanted. They proposed changes and improvements to the initial requirements to better suit their needs.	8
8	Reuse of modules without proper analysis.	Codes from a previous project were re-used for similar functions. For example, the icon for ‘Loan’ module appearing in menu when this module has not been requested by client but was present in the software.	1
9	Rework on same module numerous times.	Due to incomplete regression testing, several bugs emerging from correction of previous errors cropped up. Those modules had to be reworked.	3
10	Incomplete testing due to schedule pressure.	There were certain deficiencies in testing such as reports parameters accepting wrong input, forms accepting wrong format/range of dates as well as list of values returning empty.	4
11	Incomplete correction of bugs.	Some bugs which were corrected had an impact on other modules. The impact was not assessed properly and testing was not done thoroughly. Other issues cropped up which were a direct consequence of this. Also, corresponding documents were not updated as was supposed.	10
12	Lack of testers.	In some functionalities, bugs such as errors in data input format have been logged. These errors were left out during testing which was not done correctly due to lack of testers in the team.	2
13	Schedule estimated by project managers only.	Since developers were not consulted for their opinions on schedule, the consequence was wrong estimations were made for the software project. For example, coding the processing of payroll module took nearly twice the estimated time to reach completion.	1
14	Client not satisfied with how requirement turned out.	Clients specified their requirements according to their perspective and at the end of the software development life cycle, there were poor client satisfaction about content and layout of some reports. For example, report on passage benefits had to be reworked.	1
15	Developer not familiar with business domain of client.	Some particular business domains were difficult to understand for some developers. For example PAYE (Pay As You Earn) procedures and the processing of bonus procedures and refund of leaves procedures were difficult to understand for some developers.	3

Table 2. Root cause of logged issues (Step 2)

No	Problems	Root Cause	Score
1	Client logged request since a long time and request was serviced only later.	People	5
2	Staff lacking experience was assigned for bug fixing.	People	4
3	Absence of some requirements in SRS.	Requirements Problem	7
4	Incomplete requirements.	Requirements Problem	3
5	Misinterpretation of requirements by developers.	Requirements Problem	6
6	Misunderstanding between client and developers.	Requirements Problem	3
7	Change in requirements after coding started.	Requirements Problem	8
8	Reuse of modules without proper analysis.	Requirements Problem	1
9	Rework on same module numerous times.	Testing	3
10	Incomplete testing due to lack of time.	Unrealistic Schedules	4
11	Incomplete correction of bugs.	Testing	10
12	Lack of testers.	Testing	2
13	Schedule estimated by project managers only.	Unrealistic Schedules	1
14	Client not satisfied with how requirement turned out.	Requirements Problem	1
15	Developer not familiar with business domain of client.	Requirements Problem	3

Table 3. Total Score per cause (Step 3)

Category	Total
Requirements Problems	32
Unrealistic Schedules	5
Testing	15
People	9

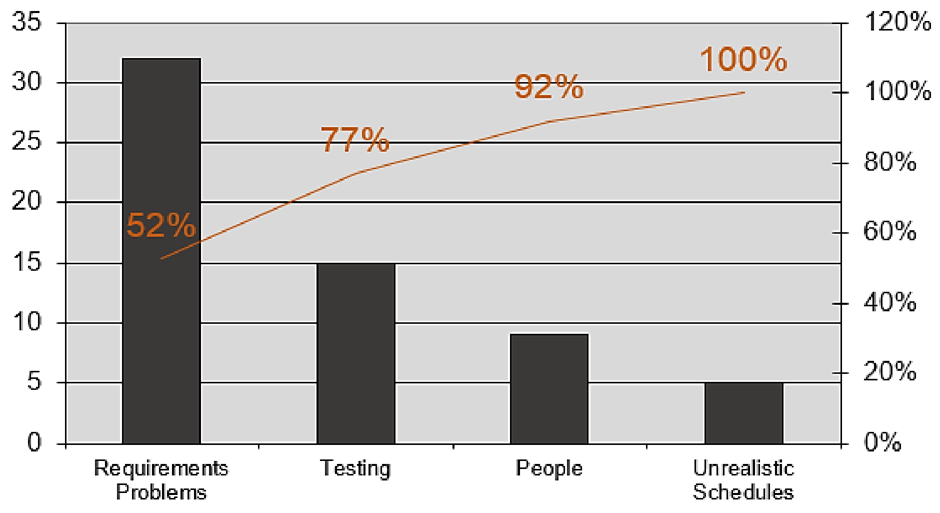

Fig 3: Pareto Chart showing the major cause of software quality problem (Step 4)

As demonstrated by the Pareto chart in Figure 3, the major cause of poor software quality is essentially issues associated with requirements. According to the chart, 52% of problems are related to requirements. Therefore, by improving the Requirements Engineering phase and focusing efforts on reducing the difficulties associated, the quality of the software produced by the company considered in this paper can be greatly improved.

Also, as mentioned earlier, requirements are gathered by developers who are not properly versed with the various techniques of requirements elicitation procedures. There is no proper framework to follow in order to collect requirements either. Requirements form the baseline for any project and must, therefore, be given utmost attention as correcting errors at a later stage in the life cycle implies a larger cost to the company. If the set of requirements are nearly perfect in the beginning of the software development life cycle itself, this will undoubtedly improve quality as well as save in terms of budget, resources and schedule. This will definitely enable a greater probability of success for the software project.

4. PROPOSED SOLUTIONS

This section proposes solutions and methods that can be used by the company to alleviate the problem of quality caused by the shortages that currently plague the Requirement Engineering process in the company considered.

4.1 Capability Maturity Model Integration (CMMI)

CMMI is a collection of best practices to improve product quality and development efficiency for both hardware and software. It is built upon three key concepts which are process areas, goals and practices. According to the Software Engineering Institute (SEI), CMMI helps with the integration of separated organizational functions, it sets process improvement goals, guide quality processes and gives a point of reference for evaluating current processes. CMMI identifies 25 so-called process areas in the development process. Each process area defines a set of so-called specific goals and a set of specific practices that serve to fulfil the goals [10].

Depending on the CMMI areas of interest (acquisition, services, development) used, the process areas it contains will vary [13]. Process areas are the areas that will be covered by the organization's processes.

The company could compare the current process being used for requirements management against the best practices proposed by CMMI, with a focus on the requirement engineering best practices. Thus, areas where improvement can be made need to be identified and worked on. Table 4 summarizes the benefits and drawbacks of the CMMI model in order to alleviate the requirement engineering problems faced by the company.

Table 4. Benefits & Drawbacks of Capability Maturity Model Integration (CMMI) for solving requirement problems

Benefits	Drawbacks
Cost saving and increases performance.	Requires a considerable amount of time and effort to put in place in the organization. It is also costly.
Helps to have stakeholders committed to requirements so that there are minimal changes in requirements.	Difficult to move back and correct requirements.
Manages changes in requirements by making use of the requirements impact analysis technique, and the change log technique.	Difficult to spot missing requirements.
Makes use of traceability matrix for requirements.	Requires much documentation that is very time consuming.
Ensures quality from conception to delivery and maintenance by maintaining alignment between requirements and project work.	Only help if implemented at an early stage.

4.2 ISO 9001 Standard

ISO 9001 are procedures that cover key processes in the software development life cycle. It monitors processes to ensure their effectiveness, check for defects on outputs, review individual processes regularly and facilitates continual improvement. The ISO 9001 standard is made up of a framework for managing an organization's processes in order to meet client expectations and to provide a consistent service and that quality is consistently improved. As control and verification mechanism, the ISO 9001 standard uses document control. Table 5 summarizes the benefits and drawbacks of the ISO9001 standard in order to alleviate the requirement engineering problems faced by the company considered in this paper.

Table 5. Benefits & Drawbacks of ISO 9001 Standard for solving requirement problems

Benefits	Drawbacks
Ensures software product meets customer requirements.	Does not cater for major changes in requirements that may occur.
Ensures conformity to applicable statutory and regulatory requirements.	Does not consider existing systems.
Improves communication.	ISO 9001 registration need heavy document workload.
Uses document control to demonstrate effective operation of the quality management system.	ISO 9000 registration process is a lengthy process as well as costly.

4.3 Formal Inspection

An inspection is a powerful tool that can help achieve significant improvements in software quality [11]. An inspection is a rigorous process and in-depth technical review to identify problems as close as to their point of origin. This process can have a significant impact towards the improvement of software quality. For example, inspections held on Motorola's Iridium project detected 80% of the defects present, whereas less formal reviews discovered only 60% of the defects [16]. Formal inspections contribute to high defect removal efficiencies. Research has shown that formal inspection is one of the common practices performed by those companies considered to be "Best in Class" globally [17].

The aims of the inspection process are to:

(A) Find problems at the earliest possible point in the software development process;

(B) Ensure that agreement is reached on rework that may need to be done;

(C) Verify that any rework done meets predefined criteria.

Table 6 summarizes the benefits and drawbacks of formal inspection in order to alleviate the requirement engineering problems faced by the company considered in this paper.

Table 6. Benefits & Drawbacks of Formal Inspection for solving requirement problems

Benefits	Drawbacks
Problems are found at the earliest point in the software development process.	Requirements are not traced.
In-depth review of requirements; ensuring customer satisfaction.	Does not consider existing systems.
Removes ambiguity in requirements.	Does not cater for major changes in requirements that may occur.
Ensures software product has less defects; more effective than testing.	Time consuming.
Caters for any possible rework to be done.	

4.4 Walkthroughs

Requirement walkthrough is an unstructured meeting where requirements documents are reviewed in order to finalize or baseline the requirements before they are handed off to the development team [14]. It will be the responsibility of the project manager to ensure that a walkthrough is performed at least twice during project planning. The project's stakeholders, the development team and the testers as well need to participate in the meetings to ensure that everyone understands the requirements. The walkthrough will help bridge the communication gap between the different stakeholders. Agreement on the requirements will ensure that the right product will be delivered and also avoid confusion at later stages in the life cycle. Table 7 summarizes the benefits and drawbacks of walkthroughs in order to alleviate the requirement engineering problems faced by the company considered in this paper.

Table 7. Benefits & Drawbacks of Walkthroughs for solving requirement problems

Benefits	Drawbacks
Focuses on finding defects.	Does not consider existing systems.
The product is looked into step-by-step.	Does not cater for major changes in requirements that may occur.
Faster turnaround.	Walkthroughs are successful only when right people are involved.
	Is a lengthy process.

4.5 Prototyping

A prototype is an initial version of a software system which may be used for experimentation and gather feedback from the customer and stakeholders of the software system. Prototyping is a flexible methodology that accommodates changes until the software is finalized and the customer is satisfied. Prototypes are valuable for requirements elicitation because stakeholders can experiment with the system and point out its strengths and weaknesses right from the beginning [15]. The use of this methodology can help the company concerned to eliminate problems such misunderstandings, misinterpretation of requirements along with incomplete requirements by presenting the stakeholders with a prototype in the early stages of the software development life cycle.

The advantages of using prototype is that it establishes feasibility and usefulness of the product before high development costs are incurred, forces a detailed study of the requirements which reveals inconsistencies and omissions and also, minimizes misunderstanding and omissions. Table 8 summarizes the benefits and drawbacks of prototyping in order to alleviate the requirement engineering problems faced by the company considered in this paper.

Table 8. Benefits & Drawbacks of Prototyping for solving requirement problems

Benefits	Drawbacks
Allows client to get a feel of the overall functionality of the product.	Once a prototype is considered as accepted, requirements cannot be changed further.
Given feedback from client, changes are made rapidly.	Many details are not built in a prototype so it is an incomplete problem analysis.
Ultimately, accurate requirements are captured.	This technique may increase the complexity of the system.
Missing functionality can be identified easily.	Too much effort may be invested if not monitored properly.

5. RECOMMENDATION: REQUIREMENT INSPECTION

Considering CMMI model, it tends to make an organization look at each level as a target. Moreover, CMMI does not outline a particular way to reach to the next level considering the requirement engineering phase of software development. It tend to focus on management related issues rather than improving software quality. It is also costly to consult CMMI professionals to become CMMI level certified, which is also a major drawback.

Achieving ISO 9001 certification can also be a very costly process, especially for small firms. Moreover, the ISO certification relies heavily on documentation and procedures that will demand more recruitment and training. In addition, studies have shown that the registration process for ISO standard is very time consuming.

Walkthroughs differ significantly from inspections because the author takes the dominant role and conducts the meeting where other specific review roles are usually not defined. Walkthroughs are informal because they typically do not follow a defined procedure, do not specify entry and exit criteria, require no management reporting, and generate no metrics [18]. Thus, walkthroughs is not an effective solution for the company considered in the paper.

Rapid development of prototypes is essential so that they are available early in the elicitation process. For the company considered in this paper, it will not be feasible to develop prototypes rapidly due to the lack of resources such as the number of developers in the team. The expense of implementing a prototype for each project may not be economically viable for the company considered.

Requirement inspection should be the one inspection that is never skipped [14]. Each major defect found by inspection saves an average of nine labor hours in avoided downstream rework [19]. In the requirements gathering process, requirements are collected and documented as detailed software requirements. The SRS is the document to be inspected in this proposed solution. This is known as R1 inspection. A requirements inspection ensures that specifications are well-written; each requirement in the SRS is consistent, accurate, unambiguous, clear, traceable and testable, that are also the software quality attributes that are used to determine the quality of software [23]. The insights gained after an inspection is carried out permit the remaining part of the work to be done in a better way. Moreover, the company will not have to invest on more resources, instead a quality plan will be set up which can be followed to conduct the inspection. Inspection can thus prove to be a powerful technique for quality improvement. The recommended solution is therefore to set up requirement inspection in the company considered in this paper.

6. PROPOSED QUALITY PLAN FOR REQUIREMENT INSPECTION

A quality plan describes the activities to be performed by an organization with the aim to achieve quality in its product. In order to conduct the recommended solution in the company discussed in the paper, that is conduct requirement inspection, the following quality plan is proposed.

6.1 Setting up of the Inspection Team

One of the most important aspects of the inspection process is to define the role of the team members which comprises of three to six people who play the roles of moderator, producer, reader, recorder and inspector in this process.

6.2 The Requirements Inspection Process

6.2.1 Planning Phase

During the planning phase, the moderator selects inspectors from different disciplines and functions in the organization. This will ensure that the SRS is inspected from many different points of view.

The important steps of the planning phase are:

(A) The software to be inspected and the inspection team are determined.
(B) Specific work products are identified for inspection from the SRS document.
(C) Selection of all the elements important for carrying out the inspection are gathered such as the identification of team members and the selection of the moderator. Moreover, the team members should follow an inspection training process prior to conducting the inspection [25].

6.2.2 Preparation Phase

During the preparation phase, the inspection meeting is prepared by critically reviewing the SRS and the work product. Once the materials are ready, they are distributed to the inspection team members so that they become familiar with the documentations. Once each team member is ready for the inspection meeting, they move to the next phase which is the formal inspection meeting [25].

6.2.3 Formal Inspection Meeting

The aim of the inspection meeting is to identify all discrepancies in the work products as compared to the SRS document. Once the preparation phase is over and all the team members are ready, the moderator calls for the inspection meeting. The moderator will be the one responsible to go around the table and solicits any potential errors or defects from the team. Each potential error or defect is discussed, and the team reaches consensus as to whether a potential problem should be recorded as an error or a defect.

(A) Each potential problem is recorded on an Inspection Problem Report form for consistency.
(B) The recorder ensures that the information entered on the Inspection Problem Report forms is complete and accurate and reflects any team discussions and clarifications.
(C) The recorder records the meeting duration information on the Inspection Process Summary Report form.
(D) The moderator decides when the meeting is over [25].

6.2.4 Follow-up

The moderator works with the team members to resolve discrepancies rose at the inspection meeting. Upon successful completion, the moderator will complete the Corrective Action portion of the Inspection Summary Form to indicate that the inspection has been successfully completed [25].

6.3 Proposed Inspection Metric

It is difficult to monitor and analyze any inspection without measuring it. To be able to plan, monitor and improve the requirements inspection process, the Fagan's Metric is being proposed [20].

Fagan introduced the Error Detection Efficiency metric M_f for measuring inspection efficiency. This metric is widely recommended for software quality improvement that can be used to measure the effectiveness of requirement inspection in the company concerned. The formula of the metric is as follows:

$$M_f = \frac{\text{Number of Defects found during inspection } (DEF_r)}{\text{The Total Number of Defects in the Product before Inspection } (DEF_{total})}$$

7. CONCLUSION

In any software projects, several issues crop up which directly or indirectly affect the quality of software. Software quality should not be compromised as it demonstrates how far the requirements, features and characteristics of the software have been completed and whether customer satisfaction has been achieved. What govern software are essentially requirements. Requirements gathering are a collaborative decision-making activity involving users, developers, customers and any other stakeholders. This paper has shown that for the state owned IT organization in Mauritius, where there are many shortcomings concerning the requirement engineering phase, which includes requirements gathering, elicitation and management can be improved via the proposed recommendation "Requirements Inspection". The latter can prove to be an effective solution as many organizations have employed inspection as a tool to detect defects and improve quality in their software worldwide. Based on this, a quality plan for the organization to achieve quality in its software products has been proposed. The quality plan describes steps and activities to be performed in order to proceed with the Formal Inspection process in the Requirement phase of the Software Development Life Cycle. The latter may at start consume a significant amount of time but the benefits to be reaped outweigh the amount of time that will be spent on the inspection process.

By conducting formal inspection on requirements, the company will thus be able to come up with clear, unambiguous, complete and verifiable requirements. This will not only save maintenance and rework costs for the company but also improve quality by a substantial margin and foster a quality culture in the company. Furthermore, the impact of good requirement engineering will be felt all along the software development life cycle.

8. REFERENCES

[1] Nuseibeh, B. and Easterbrook, S. 2000. Requirements engineering: a roadmap. In Proc. of the Int. Conf. on Soft. Eng. (ICSE), pages 35–46.

[2] Parnas, D. L. 2007. Software engineering programmes are not computer science programmes. Ann. Soft. Eng., 6(1):19– 37, 1999.

[3] Hofmann, H.F. and Franz, L. 2001. Requirements Engineering as a Success Factor in Software Projects. IEEE Software, pp. 58-66

[4] Sharma, A. and Kushwaha, D.S. 2010. A Complexity measure based on Requirement Engineering Document. Journal of Computer Science and Engineering, V1.1.

[5] Defect Prevention: Reducing Costs and Enhancing Quality. 2015. Defect Prevention: Reducing Costs and Enhancing Quality. [ONLINE] Available at: http://www.isixsigma.com/industries/software-it/defect-prevention-reducing-costs-and-enhancing-quality/. [Accessed 02 November 2015].

[6] Cost of Quality | Software Testing Fundamentals. [ONLINE] Available at: http://softwaretestingfundamentals.com/cost-of-quality/. [Accessed 12 November 2015].

[7] Error Cost Escalation Through the Project Life Cycle – NASA Johnson Space Center [ONLINE] Available at: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/2010 0036670.pdf. [Accessed 12 November 2015].

[8] Boehm, B. W. 1981. Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ.

[9] Pressman, R. S. 2001. Software engineering: A practitioner's approach (5th ed.). New York: McGraw-Hill.

[10] Höggerl, M. 2006. An Introduction to CMMI and its Assessment Procedure. In Seminar for Computer Science. University of Salzburg, Department of Computer Science.

[11] Rakitin, S.R., 2001. Software Verification and Validation for Practitioners and Managers. 2nd ed. Boston London: Artech House, Inc.

[12] Walker, D. 2015. Notes on Prototyping. [ONLINE] Available at: http://fhs.mcmaster.ca/OHPToolkit/Content/TK_Prototy ping.pdf. [Accessed 09 November 2015].

[13] CMMI V1.3 Process Areas | Ben Linders. [ONLINE] Available at: http://www.benlinders.com/tools/cmmi-v1-3-process-areas/. [Accessed 12 November 2015]

[14] Requirements Walkthrough Checklist – Project Connections. [ONLINE] Available at: http://www.projectconnections.com/templates/detail/requ irements-walkthrough-meeting.html. [Accessed 12 November 2015].

[15] Kimmond, R.M. 1995. Survey into the acceptance of prototyping in software development. Proceedings from the IEEE 6th International Workshop on Rapid System Prototyping.

[16] Brown, N. 1999. High-Leverage Best Practices: What Hot Companies Are Doing to Stay Ahead. Cutter IT Journal, v12.9, pp. 4-9.

[17] Brown, N. 1996. Industrial-Strength Management Strategies. IEEE Software, v13.4, pp. 94-103.

[18] Karl, E. 2002. Peer Reviews in Software: A Practical Guide. Addison-Wesley.

[19] Gilb, T., and Dorothy, G. 1993. Software Inspection. Wokingham, England: Addison-Wesley.

[20] Fagan, M. 1976. Design and code inspections to reduce errors in program development. IBM Systems Journal, v15.3, pp.182-211.

[21] Piscataway, N.J. 1998. IEEE Guide to Software Requirements Specification, IEEE Std. 830-1998, IEEE Press.

[22] Pareto Chart Analysis (Pareto Diagram) | ASQ. [ONLINE] Available at: http://asq.org/learn-about-quality/cause-analysis-tools/overview/pareto.html. [Accessed 12 November 2015].

[23] McCall, J. A., Richards, P. K., & Walters, G. F. Factors in Software Quality. General Electric Co Sunnyvale CA, 1977.

[24] Huzooree, Geshwaree and Ramdoo, Vimla Devi. 2015. A Systematic Study on Requirement Engineering Processes and Practices in Mauritius. International Journal of Advanced Research in Computer Science and Software Engineering, v5.2, 40-46.

[25] Rakitin, S.R., 2001. Software verification and validation for practitioners and managers. Artech House, Inc.