International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 3 (2015) pp. 8203-8219

http://www.ripublication.com

Requirement Management – Controlling Quality At The Upstream In Commercial Software Project Management

Sanjay Mohapatra

Associate Professor, Information System,
Xavier Institute of Management, Bhubaneswar
Email: sanjay_mohapatra@yahoo.com, sanjaym@ximb.ac.in
Phone: +91 94370 00659, Fax: +91 674 2300995

Abstract

Requirement Management in commercial software application development not only talks of gathering requirements from the customer it also highlights the need for a scientific step by step approach for eliciting requirement. Without thorough understanding of the requirements from customer point of view at this stage will lead to a cascading effect later on as it would involve rework and extra cost. Hence a complete scientific management approach is needed for this stage which is known as requirement management. Requirement Management takes the accountability of not only meeting the technical specifications of the customers but also involves defining an approach to take care of any change in the requirement needed later on. In addition to that requirement management takes care of tracing back each and every functionality requirement to the final delivered output. In this paper, based on experience, tools and techniques that can be used to gather requirements for commercial software application development have been presented. The process needed for managing change requests have also been discussed. Using this approach, the author has been successful in meeting customer requirements and reducing rework effort due to less defects being injected during project life cycle.

Subject Area & Key Words: Requirement Management, commercial software, traceability

Introduction

Requirement engineering is an age old topic and is a starting point for any software

project. It is an important component in effective software engineering. However this is the most difficult tasks to be carried out (Brooks, 1987; Berry et al, 2005). A good requirement management at the beginning of the software engineering is thus a necessity for executing projects successfully (Brooks, 1987; Procaccino et al., 2002). This improves quality of deliverables, reduces risks associated with software development. The study conducted by Standish group indicated that there has been a failure of 78 percent of projects because of ineffective requirement management (www.Standishgroup.com, 2003). This failure has been attributed to lack of understanding of requirements, unclear requirement input, incomplete and changing requirements. If the requirements are not understood properly or if not recorded correctly, will definitely lead to serious issues later in project which will lead to higher cost, rework, and late delivery of the software. This also has a potential risk of dissatisfied customer. However this study is limited to past projects and does not relate to (i) commercial software and (ii) does not talk of practices that can be followed to improve requirement management. A good requirement management is necessary to prevent cascading effect later on. Controlling quality at requirement stage will help us see a good quality of output at the end. As a result requirement management assumes an important role. In a study Kaindl et al., 2002 indicated that even though it has been accepted that requirement management plays an important role in the success of software application development, still available literature does not talk about a systematic approach for requirement management.

The participants in this stage also play a great role in successful execution of the project later on. To understand customer's requirement, it is imperative that the person gathering requirements (called analyst) is knowledgeable in the domain area that the customer is working in. He should be able to grasp the business process that the customer is following and should be able to see a higher picture of the customer's approach to business with respect to domain knowledge. In addition the analyst should be comfortable with technical concepts relating to hardware and software, language and any such relevance to system being developed. Also familiarity with any particular tool used by the customer will be necessary.

As evident above there are some important activities at the requirement stage and hence there is a need for a systematic approach. In this paper the author has given a real life experience. The paper is based on experience that the author has in implementing various projects and the author has given various issues that he had faced and the systematic approach that he has been choosing in requirement management.

Literature Review

Studies have indicated that software quality, project management and software development activities are positively influenced by good requirement management

(Berry et al, 2005; Brooks, 1987; Weigers, 2003; Stevens, 1994). Literature available from studies (Opdahl, 1994; Mar, 1994; and Macaulay, 1996) reveal that efforts spent on structured requirement management at the beginning of the project will reduce defects being injected during the project execution life cycle thus improving quality of deliverables. Different literatures available through studies conducted by Yu (1997), Spanoudakis et al. (1997), Playle et al. (1996), Krogstie et al. (1995), Kosman (1997) have shown improvements made in reducing risks associated with software projects, reduction in rework effort, reduction in delivered defects at the customer site through improved requirement management. These improvements have been possible through different initiatives carried out by these researchers in their respective studies. This improvement in quality and other aspects of software engineering are also possible for a product development organization (Vartiainen et al., 2004; Kauppinen et al., 2002; Kujala et al., 2001). These improvements were achieved through different initiatives carried out to improve effectiveness of requirement management. Improvement in requirement management also had effects on the culture (also reported by Holtzblatt, 1995), supporting processes and technology and the level of motivation among the employees involved in project execution. One of the improvement initiatives taken by Jacobs (1999) emphasized on breaking requirements into smaller requirements. This helped in understanding requirements into smaller comprehensible requirements which could then be easily converted to specifications by designers. This also helped in understanding customer requirements and improved communication with customers. Subsequent reviews and defect detections became easier in the downstream life cycle stages. This inference has also been drawn by Wyder (1996), Wiley (1999), Robinson et al. (1994), Regnell (1995), and Brien (1996) where these researchers have used use cases to break requirements into smaller use cases and then develop the applications based on use cases.

The conclusion drawn from these studies is that requirement management plays a vital role in software applications. In the case of commercial applications, it plays a larger role as defects and effort spent on inspection or formal reviews and subsequent reworks extends completion of timeline for completion. These results in customer dissatisfaction, delay in final delivery and also quality of the deliverables suffer. In this paper an attempt has been made to demonstrate tools and techniques that would force the project team to follow a systematic approach to gather requirements from customer.

Requirement Management

Requirement Management consists of two major activities: gathering requirement analysis and specifications and requirement change management (www.standishgroup.com). Gathering requirement specifications are done at the start of any project where as change management is done through out the project. A

related sub activity for gathering requirement specification is defining requirement traceability (MacFarland, 1995). Requirement traceability ensures that all specifications and functional requirements gathered at the start have a direct relationship with the final design and code being delivered. As a result we can always say that the final software always meet the requirements as desired by the customer (Ramesh, 1995; Gotel et al., 1997). And this process also takes care of any change in requirement as desired by the customer at any stage of the project. This will be explained later on in another section.

Gathering Requirement Specification

The output of this stage is to produce a document that will contain all requirements that the final software will have. Usually this document is termed as System Requirement Specification (SRS) even though the nomenclature can vary depending on naming convention followed by the customer.

The steps followed are:

- Gather requirements and analysis
- Documenting requirements
- Review requirements
- Obtain customer sign-off for SRS

The activities shown above look quite simple but in reality they need a lot of preparation and deal with not only domain expertise but also technical knowledge and inter personal skill. Since an analyst needs to go through a lot of preparation and then finally come up with desired output, it is highly recommended a checklist be prepared before hand which will guide the analyst through requirement gathering phase. The checklist explained below is the checklist that the author has developed based on his own experience in this field over last 14 years. However one can modify this checklist to suit one's needs. The pre assumption of this model is that the analyst has expertise in domain knowledge.

Requirements Analysis Activity Checklist

Understand the present business model – As Is business model

Discuss with various users/customers to get a grasp on the present business model. Even if the analyst has expertise in domain knowledge it is imperative that he understands the present business model that the customer has adopted. Hence draw the present business flow as understood by the analyst and present the model to the user. This model should have all entities, their relationship with each other and if possible map to organization structure. This presentation should be backed up with data. Above presentation in a pictorial form is recommended as this will bridge

communication gap, if any.

Once the user accepts the presentation the analyst should go for defining business requirements. In this phase the objective is to develop To-Be business process model. This model should contain requirements/functionalities as desired by the customer. If required all required changes with implementation time frame be noted down.

A checklist is shown below to explain the process described for To-Be model.

Develop To-Be business process model

Review any documented future business requirements
Create To-Be business process flows
Define integration points of processes
Obtain end-user agreement on the To-Be Process model
Identify changes in processes
Define detailed requirements
Determine future business requirements
Determine technical architecture requirements
Determine audit and control requirements
Determine data security requirements

Define To-Be data model

Define key data entities by application
Define data access by entity for each organization type
Define data timeliness requirements for major data entities shared by different organizations
Define candidate data registries and shared entities
Define target data flows between business functions
Gather volumes and frequencies for data flows

After defining To-Be model, perform gap analysis. This analysis will consist of finding gaps between the present system and the required system. This analysis will also try to find the feasibility of adopting available hardware and software configuration to the required To-Be model. At this point of time the analyst needs to discuss with the customer regarding the budget for the new system as that will determine conversion and integration feasibility of available hardware and software configuration to the desired functionalities. However there would be various issues related to gathering requirements. These are explained in the next section.

ISSUES in requirements elicitation

Attention to the following issues is recommended, during the preparation of the Requirements Specification document

i. Deciding a time frame between current and future needs
ii. Assessing future requirements for building it into the design
iii. Ensuring participation from key members on the customer's side, as their normal work takes priority. It is recommended that persons from the customer IS department and end user group are involved as a team during requirement analysis.
iv. Conducting comprehensive interviews covering boundary conditions, error conditions, manual and automatic procedures, and information flow in the system.
v. Handling conflicting customer needs and getting a clarification in such cases without any prejudice.
vi. Resolving pending issues that depend on policy decisions by the customer
vii. Methods for agreeing on requirements and approving changes
viii. Anticipating enhancements and future life-cycle issues
ix. Anticipating maintainability/portability issues in application's life-cycle
x. Pro-actively consult with latent and important opportunities from technology (e.g. Client/server) to the customer's business.

Underlying Difficulties of Requirements Elicitation

The author has experience in doing a detailed requirement study across different countries as well across different cultures. These requirements were also carried out from different language speaking groups. He (the author) has worked for customers from India, from USA, Ireland and China. As a result he has experienced cultural as well as language issues while eliciting requirements. However a scientific approach has always helped in gathering requirements and implementing the project successfully. Hence the following points will help in minimizing error at the time of eliciting requirements.

Articulation Problems

- These include problems both the user's expression of needs and the analyst's understanding.
- The users may be aware of their needs, but they are neither able to articulate them nor understand how technology can help them. Even they may be afraid to articulate it. This behaviour is commonly seen while executing projects in Far East (including China) and in India.
- There could be a misunderstanding in the concepts and terminology between the users and the designers.
- No single person has the complete picture. This is especially true for complex systems where each user may have only a limited perspective of the system to be built.
- The analyst may have preconceived idea about the system being followed. As

a result he fails to appreciate all information that the user is providing. This usually happens when the designers believe that they have already understood the user's needs.

Communication Barriers

- The users and the developers come from different worlds and have different vocabularies. This is particularly true for software applications being outsourced to India from US and Europe.
- Problems due to the medium of communication.
- Incompatible styles of interaction can lead to a breakdown of communication. (e.g. some are assertive, some are not assertive, some deal with details and some with abstraction)
- There are different personality types and different value systems among people.
- A typical problem occurs while reading body language. Where as a person from USA nods his head in agreement to a point being discussed, a person from Far East will shake his head for saying 'yes'. Differences in body language sometimes lead to wrong conclusion from the analyst.

Knowledge and Cognitive Limitations

- The team of users and analysts don't have adequate domain knowledge, so make wrong decisions.
- No person has perfect memory. Further more, the oral and written communication may be misinterpreted.
- People sometimes have difficulty with the scale and complexity of the system. Some try to simplify the problem, but not always in the valid way.
- The users tend to state the problem in terms of the favoured solution.

Human Behaviour Issues

- There are sometimes conflicts and ambiguities in the roles that the users and the customer IS department play in the elicitation process.
- There could be a fear that the new system would result in automation which would lead to retrenchment in the future.
- The development of the software system may result in a fear in individuals that the system will necessitate all kinds of changes in behaviour of individuals and groups.

Review Requirements Analysis Document with the User Group

Once the requirement elicitation is complete, System Requirement Specification (SRS) document is prepared and reviewed. Review of SRS is important as the analyst needs to take care of proposed architecture, identify required custom support systems, judge if there could be any issues with the performance of the system and

finally to make business transition smooth if there is a transition from present business model to a new one.

Propose Architecture

Develop conceptual application and database architecture
Identify business policies affecting the application architecture
Schedule and conduct meetings on critical application decisions
Define the baseline to target application replacement mapping
Define the conceptual application architecture
Define the proposed enterprise support systems
Define the conceptual database architecture

Identify required Custom Support Systems

Determine requirements for enterprise reporting and decision support systems
Identify other sub systems required for production

Assess performance risks

Identify critical processing requirements by the Organization
Gather throughput capability information for applications and hardware
Initiate performance assurance tests if necessary
Identify all perceived risks on performance
Determine business transition requirements
Determine the resources, business systems, software and hardware to be migrated
Determine the migration priority and sequence for resources and business systems
Determine implementation contingency situations
Define an initial transition plan
Decide on the training needs of the users for this transition
At the end obtain approval for Requirements Analysis Document

A sign-off from the customer is needed to start on the next phase of work.

A sample checklist for requirement gathering is shown below:

Project Code:		
Subsystem :		
Doc Number : Version of requirements		
Date of review :		
Prepared By:		
Reviewed By :	**Checked Y/N**	
Checklist		**Remarks**
Check whether the system objectives are stated clearly		
Check whether the system boundaries are clearly specified		
Check whether key business events are clearly listed		
Check whether inputs and outputs to each event described clearly		
Check whether required precedence relationships among events a e clearly stated		
Check whether all screens are clearly specified		
Check whether standards for reports are clearly stated		
Check whether periodicity of report generation has been mentioned		
Check if security and access control to reports and screens have been defined		
Check if layout has been accepted by the user		
Check if delivery to printer is needed		
integrating external print matter is needed		
Publishing (Style sheets, graphics) is accepted by end user		
Distribution of reports has been defined		
Check whether interfaces with other systems are described		
Check whether the required hardware is described		
Check whether specific software requirements are documented		
Is there a need for any other software including networking software		
Check whether networking Issues and connectivity requirements are documented		
Check whether specific performance requirements, if any, are documented		
Check whether user interface standards, if available, are documented		
Check whether detailed design standards, if available, are documented		
Check if coding standards, if available, are documented		
Check if document standards, if available, are documented		

Check whether the required security checks are documented		
Check whether system availability requirements are documented with respect to following:		
Seasonal requirement		
acceptable down time limits		
Reliability		
Transaction Volume &Data Volume		
Backup &Recovery		
Is Data Migration needed?		
Are the constraints/issues to data migration been defined		
Check whether any possible design constraints/standards are documented		
Specific recommendations or Concurrency issues have been mentioned		
Build up of referential integrity through programs of using possible facilities in the		
Check if prototyping requirements, if any, are documented		
Check whether requirements are traceable through downstream processes		
Requirements are numbered		
Requirements are stated such that changes can be tracked		

REQUIREMENTS CHANGE MANAGEMENT PROCESS

Need for change Management

Requirements will change. This is normal, frequent, and good! The only systems for which requirements will not change are those that have no users, no customers, and no stakeholders. The fact that requirements are changing on your project means that people care. There are three tricks to managing the inevitable changes to requirements: (1) reduce the development life cycle time, (2) maintain lists of requirements, and (3) handle each change in an intelligent manner.

Requirements do not remain frozen but change during the course of the project. Hence proper methods for handling requirement changes are essential. In this paper the author suggests a change management procedure to record all the change requests.

This procedure is used after the analyst has received sign-off from the customer. Any request for a change in the requirement after sign-off will be recorded as change request. As soon as a change request is received it will be first logged. A detailed impact analysis is carried out at this stage. This analysis will calculate the extra effort needed to accommodate this change request. In addition it

will also find the possible impact on the schedule – i.e. if there could be a delay in delivery the final output that was promised to the customer. This analysis then needs an approval from the customer as the change request has an implication on initial estimates on cost as well as delivery schedule. All such change requests are then maintained in a spreadsheet which can be used to find the overall effect all change requests received.

Requirements change management log template

The following is a simple change request log template for change specification. This is used to track the entire change request received. More elaborate templates can be designed by individual projects according to their specific needs.

The simplicity of the template makes it suited for deployment as an Excel spreadsheet.

Change Request#	Request By	Request raised date	Function	Offshore Resp	Onsite Resp	Total Effort (in Person Hrs)	Start Date	End Date	Finish By Date	Status	Remarks

Notes:

1. A change request can fall into one or more categories such as Business changes, Design changes, Schedule changes, Budget changes, or Contract changes.
2. Each change request can typically be in one of the five following status: Pending, Approved, Rejected, Work in Progress, Completed or Abandoned.
3. It is recommended that the "Impact" clause be exploded to indicate the specific work products that will be impacted, and the effort involved for each. When this is done, it becomes easier to track the changes to each of these work products.

In performing an impact analysis, it is convenient to have a detail checklist of work product types (such as programs, detailed design specs, UI specs, etc.) that need to be run through. Such a checklist is best arrived at by looking at the list of items under configuration control, as documented in the configuration management plan of the project.

Traceability Management

The basic objective a software project is to deliver a software that would match functionalities as desired by the customer. Hence it is imperative that there should

be a mechanism to check or trace back these requirements to the delivered software. That means we should be able to link each of requirement to the work product that is being delivered (entity name in the sample below). This will also help design test cases needed for the project. This kind of tracing will also validate that each and every requirement has been met and also has been tested for all functionalities. In their study, Lalioti et al. (1995), Bruno et al. (1995), Lultz et al. (1996) and Rosenberg (1998) consider this approach as a validation technique used for making sure that all the requirements have been converted into design specifications and developed by the project team.

There could be two types of traceability – forward and backward (www.sei.cmu.edu). In forward traceability as shown in the sample below each and every requirement is traced to the final output where as in backward traceability the output is traced back to the original requirement. Backward traceability is needed for debugging during integration and final phase of testing.

A sample traceability matrix

Requirement #	**Functional Specification**	**FuncSpec Section**		**Location**	**Entity Name**	**Type**	**Status**
				FrmMDIMain		VB	Delivered
				Global.bas		VB	Delivered
				Mainline.bas		VB	Delivered
				FrmSeparatePerson		VB	Delivered
				NonAgtSearchResults.cls		VB	Delivered
1	Requirement #1	4.1	NonAgent	NonAgent.cls		VB	Delivered
				PersonWinsign.cls		VB	Delivered
				PesronWinsigns.cls		VB	Delivered
				AdPersonForSeparateByAgyRet		TSQL	Delivered
				AdPersonForSeparateByNameRet		TSQL	Delivered
				AdEntrprsSecurityUpdate		TSQL	Delivered
				CombinePeople.frm		VB	Delivered
				NonAgent.cls		VB	Delivered
				NonAgentSearchResult.cls		VB	Delivered
				NonAgentSearchResults.cls		VB	Delivered
2	Requi	4.	NonA	AdPersonForMergeByAgyRet		TSQL	Delivered

	reme nt #2	2	gent			
				AdPersonForMergeByNameRet	TSQL	Delivered
				AdMergeNonAgent	TSQL	Delivered
				FrmMDIMain	VB	Delivered
3	Requi reme nt #3	4. 3	NonA gent	Global.bas	VB	Delivered
				Mainline.bas	VB	Delivered
				PersonWinsign.cls	VB	Delivered
				PesronWinsigns.cls	VB	Delivered
4	Requi reme nt #4	4. 4	NonA gent	Person.cls	VB	Delivered
				NonAgentView.frm	VB	Delivered
				PrintNonAgtPersonalInfo	VB	Delivered
				AdEntrprsSecurityRetrieve	TSQL	Delivered
				NonAgentView.frm	VB	Delivered
				Telephone.cls	VB	Delivered
				Telephones.cls	VB	Delivered
				AgentView.frm	VB	Delivered
				CustomizeAgentView.frm	VB	Delivered
				PrintNonAgtPhone.bas	VB	Delivered
				PrintLocation.bas	VB	Delivered
				PrintPhone.bas	VB	Delivered
				PrintAddress.bas	VB	Delivered
5	Requi reme nt #5, 6	4. 5	NonA gent	PrintOptions.frm	VB	Delivered
		4. 6		FIMS.INI	VB	Delivered
				Main.bas	VB	Delivered
				TelephoneAddChange.frm	VB	Delivered
				AdTelephoneUpdate	TSQL	Delivered
				AdTelephoneRetrieve	TSQL	Delivered
				AdTelephoneInsert	TSQL	Delivered
				AdLocationContextRetrieve	TSQL	Delivered
				AdLocationContextInsert	TSQL	Delivered
				AdLocationContextUpdate	TSQL	Delivered
				AdLocationInsert	TSQL	Delivered

The description in the above table is self explanatory. Functional name refers to the requirement number in SRS document where as the entity name refers to the name of the final work product. Function name refers to the clause/section number in SRS.

Benefits

Benefits of following the tools and techniques, as mentioned above, have been evident for customers as well as for the team engaged in developing software. While customer got his required deliverables with less defects, the team spent less time in reworking requests as well as on defects. Customers could meet the business objectives of their organizations which deployed the software. This also helped them to use the error free software deliverables as a strategic selling proposition and gained higher market share. The project team, on the other hand, had to spend less time in reworking defects and hence their productivity, moral of employees and work life balance improved. Thus, the tools and techniques descried here can be practiced in other similar organizations engaged in developing software in IT industry.

Conclusion

Software quality implies that the software is reliable, maintainable, usable, and that it satisfies customer expectations, user needs, and developer goals. Requirements specifications serve as the reference point for customer expectations and user needs during and after development. The requirements specification therefore plays a crucial role in the achievement of software quality. If the requirements do not exhibit quality, it is highly unlikely that the software product based on those requirements will exhibit quality. Requirements Management is an integral part of this process wherein requirements serve as living entities which are at the center of development activity. Once elicited, requirements are documented and managed with the same degree of care that we provide to our code work products. This process puts the team in control of their project and helps them manage both the project and its scope. Lastly, actively managing changing requirements keeps the project under control and helps assure the reliable, repeatable production of high quality software products.

Requirement Management is an important activity in software development project. Since the quality and success of the project depends on this stage of the project. Hence it is recommended that experience senior and experienced resources in an organization should take up the tasks of requirement management. These senior personnel are not only are experienced in project management, they are also well versed with process or model that an organization follows. This along with their domain knowledge will help the analyst team to elicit a good response from the

customer. Their expertise in dealing with human nature and overcoming communication difference will help result in a good starting point for a software project.

References

A. Opdahl, “Requirements Engineering for Software Performance,” presented at International Workshop on Requirements Engineering: Foundations of Software Quality, 1994.

B. Mar, “Requirements for Development of Software Requirements,” presented at Fourth International Symposium on Systems Engineering, 1994.

C. Ramesh, “Implementing Requirements Traceability: A Case Study,” presented at Second International Symposium on Requirements Engineering, 1995.

1. B. Regnell et al., “Improving the Use Case Driven Approach to Requirements Engineering,” presented at Second IEEE International Symposium on Requirements Engineering, 1995.

2. B. Wiley, Essential System Requirements: A Practical Guide to Event-Driven Methods, Addison- Wesley, 1999.

D. Berry, D. Damian, A. Finkelstein, D. Gause, R. Hall, and A. Wassyng, “To Do or Not to Do: If the Requirements Engineering Payoff Is So Good, Why Aren’t More Companies Doing It?” Proc. Requirements Eng., 2005.

E. Yu, “Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering,” presented at IEEE International Symposium on Requirements Engineering, 1997.

F. Brooks, “No Silver Bullet: Essence and Accidents of Software Engineering,” Computer, vol. 20, no. 4, pp. 10-19, Apr. 1987.

G. Playle and C. Schroeder, “Software Requirements Elicitation: Problems, Tools, and Techniques,” Crosstalk: The Journal of Defense Software Engineering, vol. 9, iss. 12, December 1996, pp. 19-24.

H. Spanoudakis and A. Finkelstein, “Reconciling Requirements: A Method for Managing Interference, Inconsistency, and Conflict,” Annals of Software Engineering, vol. 3, 1997.

I. Beyer and K. Holtzblatt, “Apprenticing with the Customer,” Communications of the ACM, vol. 38, iss. 5, May 1995, pp. 45-52.

3. H. Kaindl, S. Brinkkemper, J.A. Bunenko, B. Farbey, S. Greenspan, C. Heitmeyer, J.C. Leite, N. Mead, J. Mylopolous, and J. Siddiqi, “Requirements Engineering and Technology Transfer: Obstacles, Incentives and an Improvement Agenda,” Requirements Eng. J., vol. 7, pp. 113-123, 2002.

J. McFarland and I. Reilly, “Requirements Traceability in an Integrated Development Environment,” presented at Second International Symposium on Requirements Engineering, 1995.

K. Krogstie et al., “Towards a Deeper Understanding of Quality in Requirements Engineering,” presented at Seventh International Conference on Advanced Information Systems Engineering (CAiSE ’95), 1995.

4. J. Procaccino, J. Verner, S. Overmyer, and M. Darter, “Case Study: Factors for Early Prediction of Software Development Success,” Information and Software Technology, vol. 44, pp. 53-62, 2002.

L. Holtzblatt and H. Beyer, “Requirements Gathering: The Human Factor,” Communications of the ACM, vol. 38, iss. 5, May 1995, pp. 31-32.

5. K. Weigers, Software Requirements, second ed., Microsoft Press, 2003.

M. Macaulay, Requirements Engineering, Springer-Verlag, 1996.

6. L. O’Brien, “From Use Case to Database: Implementing a Requirements Tracking System,” Software Development, vol. 4, issue. 2, February 1996, pp. 43-47.

7. L. Rosenberg, T.F. Hammer, and L.L. Huffman, “Requirements, testing and metrics,” presented at 15th Annual Pacific Northwest Software Quality Conference, 1998.

8. Mohapatra. S, Maximising productivity by controlling influencing factors in commercial software development, International Journal of Information and Communication Technology, 2011, Vol.3, Issue No. 2.

9. Mohapatra. S, Gupta. D.K, Finding factors impacting productivity in software development project using structured equation modelling, International Journal of Information Processing and Management, 2011, Vol. 2, Issue 1.

10. Mohapatra. S, Mohanty. B, Defect prevention through defect prediction: A case study at Infosys, IEEE International Conference on Software Maintenance, ICSM, 2011

11. Mohapatra. S, Improvised process for quality through quantitative project management: An experience from software development projects, International Journal of Information and Communication Technology, 2010

N. Kauppinen, S. Kujala, T. Aaltio, and L. Lehtola, “Introducing Requirements Engineering: How to Make a Cultural Change Happen in Practice,” Proc. IEEE Joint Int’l Requirements Eng. Conf. (RE ’02), pp. 43-51, 2002.

12. M. Vartiainen , M. Kauppinen, J. Kontio, S. Kujala, and R. Sulonen, “Implementing Requirements Engineering Processes throughout Organizations: Success Factors and Challenges,” Information and Software Technology, vol. 46, no. 14, pp. 937-953, 2004.

O. Gotel and A. Finkelstein, “Extending Requirements Traceability: Lessons Learned from an Industrial Case Study,” presented at IEEE International Symposium on Requirements Engineering, 1997.

13. Rath. A, Kumar. S, Mohapatra. S, Thakurta. R, Decision points for adoption cloud computing in small, medium enterprises (SMEs), International Conference for Internet Technology and Secured Transactions, ICITST 2012, 2012

14. R. Kosman, “A Two-Step Methodology to Reduce Requirements Defects,” Annals of Software Engineering, vol. 3, 1997.
15. R. Lutz and R. Woodhouse, “Contributions of SFMEA to Requirements Analysis,” presented at Second IEEE International Conference on Requirements Engineering, 1996.
16. R. Stevens, “Structured Requirements,” presented at Fourth International Symposium on Systems Engineering, 1994.
17. S. Jacobs, “Introducing Measurable Quality Requirements: A Case Study,” Proc. Fourth Int’l Symp. Requirements Eng., pp. 172-179, 1999.
18. S. Kujala and M. Kauppinen, “Starting Improvement of Requirements Engineering Processes: An Experience Report,” Proc. Third Int’l Conf. Product Focused Software Process Improvement (Profes), pp. 196-209, 2001.
19. T. Wyder, “Capturing Requirements With Use Cases,” Software Development, vol. 4, iss. 2, February 1996, pp. 36-40.
20. T.S. Group, “The CHAOS Report,” The Standish Group International, 2003, ww.standishgroup.com.
21. V. Lalioti and B. Theodoulidis, “Visual Scenarios for Validation of Requirements Specification,” presented at Seventh International Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 1995.
22. W. Robinson and S. Fickas, “Supporting Multi-Perspective Requirements Engineering,” presented at IEEE International Conference on Requirements Engineering, 1994.
23. www.sei.cmu.edu