RESEARCH PAPER | Engineering | Volume : 4 | Issue : 1 | Jan 2014 | ISSN - 2249-555X

Requirement Management Best Practices

KEYWORDS: Requirement, Specification.

Dr. Kirti Mathur

International Institute of Professional Studies, DAVV, Indore. 102, Jaora Compound, Indore

ABSTRACT *Requirements play an indispensable role in the development of software. Due to unrealistic, unachievable and incomplete requirements, consequence are the software developed from these is not robust and result into poor performance or failures, sometimes with critical effect on the environment. This paper presents the in- depth study of requirement engineering concepts, hindrances and proposes best practices to tackle with these situations.*

1. Introduction

Requirements engineering is primarily a communication activity. Miscommunication problems can occur early, if project participants have ambiguity in understanding the "requirements". As requirements are further converted into specifications for implementation, describing how the system should behave. There may be some constraints on the development process of the system. A project addresses basically three levels of requirements, which come from different sources at different stages like business requirements, functional requirements and non-functional requirements.

Lack of consensus over the entire software requirements can lead to confusion. The domain of requirements engineering can be divided into:

(a) Requirement spawning
(b) SRS management

1.1 Requirement spawning

The deliverable from requirements spawning is a baseline that constitutes an agreement among project stakeholders of the new product's capabilities. It is further subdivided into elicitation, analysis, specification, and verification.

1.1.1 Key Practices of Elicitation

Software product development must begin after gathering the different requirements from all stakeholders. The critical practices for elicitation are:

	Practice	Description
1.	Illustrating the Product's Business Requirements	Documentation of product scope, limitations and vision statement.
2.	Achieving user feedback	By: -Developing scenarios and use cases -Resolving conflicts of proposed requirements -Defining implementation priorities -Specifying quality attributes -Inspecting requirements documents -Evaluating and prioritizing enhancement and change requests.
3.	Emphasize on User Tasks through use-case	Use case helps analyst: -derive functional requirements. -Shift requirements discussions from the traditional focus on features/ functionality to what the user will do with the product. -identify exceptional conditions the system must handle instead of expected system behaviours.
4.	specify Quality Attributes	-like *how well* the system will perform its functions (non-functional requirement).

Table1: Elicitation practices

1.1.2 Key Practices for Analysis

Requirements analysis includes:

(i) Decomposing high-level requirements into functional requirements.
(ii) Constructing graphical requirements models, and prototypes.

Analysis models and prototypes provide alternative views of the requirements, which often reveal errors and conflicts that are hard to spot in a textual SRS. The practice can be:

Define Priority Projects having resource limitations must set the priorities of the requested requirements. This helps the project manager plan for staged releases, make trade-off decisions, like requests for adding more functionality. A better approach is to base priorities on some objective analysis of how to deliver the maximum product value within the schedule, budget, and staff constraints, by classifying requirements into "must" and "want". Idea should be provide the greatest value at the lowest risk and cost.

1.1.3 Key Practices of Specification

The most essential specification key practice is to write down the requirements in some accepted, structured format as to gather and analyze them. The objective of requirements *spawning* is to communicate a shared understanding of the new product among all project stakeholders. Traditionally, it is captured in the SRS written in natural language, along with appropriate analysis models.

Use Tools for Storing to store requirements in a multi-user database. These tools allow manipulation of database contents, importing, exporting requirements, and connecting requirements to objects stored in testing, design, and project management. Define attributes for each requirement, such as its version number, author, status, origin or rationale, allocated release and priority. Traceability links between individual requirements and other system elements help One evaluate the impact of changing or deleting a requirement. Web access permits real-time sharing of database updates with members of geographically distributed teams. Several widely used requirements management tools are Caliber, DOORS, Requisite Pro, RTM Workshop, Vital Link etc. [4] , these give one more control over the requirements collection.

1.1.4 Key Practices of Verification

Verification involves evaluating the correctness and completeness of the requirements, to ensure that a system built

Volume : 4 | Issue : 1 | Jan 2014 | ISSN - 2249-555X

will satisfy the users' needs. Also ensures that the requirements provide an adequate basis to proceed with design, construction, and testing. This is achieved by:

Inspecting the SRS to fix defects later in the development process, formal inspection of requirements is perhaps the best software quality practice available. It reveals many defects, incurs low cost. Combining formal inspection with incremental informal [11] requirements reviews provides quality product.

1.2 SRS Management
Requirements management activities includes, evaluating the impact of proposed changes, tracing individual requirements to downstream work products, and tracking its status during development. One can monitor project status by knowing what percentage of the allocated requirements have been implemented & verified, just implemented, or not yet fully implemented. But the idea of requirement management is:

1.2.1 Manage Change requests
Every project must establish a change control board of the decision-makers who approve or reject each proposed change. Every project should have a documented process describing, how a proposed change will be submitted, evaluated, decided, and incorporated into the requirements baseline. One can support the change control process with a problem- or issue-tracking tool, but a tool is not a substitute for a documented process.

2. Engineering Practices
Process changes should be motivated by schedule slippages, overtime, rework, high repair costs, and customer dissatisfaction. The improvement-driven organization will examine the sources of such pain and avoid repeating the same problems. The problems encountered while implementing the improved requirement practices and their solutions are :

- All terminology used in requirements engineering must lie close to reality of the environment for which a machine is to be built.

Practice is give meaning to terms such that it lies close to real world and environment, by providing an informal explanation of it, which is clear and precise, written as "designations" and maintained as an essential part of the requirements documentation. The association between the physical quantities of interest and their mathematical representations must be carefully defined, and avoid the use of prose in specifications causing ambiguity.

It is not necessary to describe the machine to be built. Rather, the environment is described in two ways: as it would be without or in spite of the machine, and as we hope it will become because of the machine.

Practice is, Requirements must describe what the desired machine does, not how it does. More precisely, requirements are supposed to describe the interface between the environment and the machine, and not the machine.

- Assuming that formal descriptions focus on actions, Identify which actions are controlled by the environment, which are controlled by the machine, and which actions of the environment are shared with the machine. All types of actions are relevant to requirements engineering, and might need to be described or constrained formally. If formal descriptions focus on states, then the same basic principles apply in a slightly different form.

Practice is all statements made in the course of requirements engineering are statements about the environment. The primary distinction necessary for requirements engineering is captured by two grammatical moods. Statements in the "indicative" mood describe the environment as it is in the absence of the machine or regardless of the actions of the machine; these statements are often called "assumptions" or "domain knowledge."

Statements in the optative mood describe the environment. Optative statements are "requirements" and describe the environment.

- The primary role of domain knowledge in requirements engineering is in supporting refinement of requirements to implementable specifications. Correct specifications, with appropriate domain knowledge, imply the satisfaction of the requirements.

Practice is since requirements fully satisfy a customer. A specification is always implementable. The gap between requirements and specifications is called refinement of requirements. Specification refinement is concerned with removing the features of a specification that are not executable by the target implementation platform, and replacing them with features that are executable on that platform. Requirements refinement is concerned with identifying the aspects of a requirement that cannot be guaranteed by a computer alone, and replacing them until are fully implementable.

It has also long been recognized that domain knowledge plays an important role in requirements engineering, A specification that is implementable in principle may not be implementable in practice. Some requirements are already implementable, but some are not directly implementable reasons being:

	Reasons	Illustration
1.	Environmental Constraints	Must be verified by a demonstration specifying properties, with domain knowledge, guarantying the satisfaction of the requirement.
2.	Not-shareable information	Like in the banking environment, there is a requirement "a withdrawal request, provided that the requested amount does not exceed the current balance". The only shared phenomena in this example are the action types deposit, withdrawal-request, and withdrawal-payment. Since the balance state component exists in the environment and is not shared with the machine, the machine cannot directly implement this requirement.
3.	Forward referencing	Requirements stated in terms of future, are satisfied with the help of domain knowledge, by relating the future to the past.

Table2: Non implementation reasons.

Other aspects Emphasis on various agents within the environment is the recent trend in the study of requirements refinement. These agents often cooperate with the machine and satisfy unrefined requirements. Agent-centered refinement is a special case of refinement, where agent in the environment is indeed domain knowledge, but not all domain knowledge takes the form of agent descriptions. Some domain knowledge captures static relationships in the environment. For example, "indirect access" is a requirements refinement in which the machine needs some information it does not have direct access to; an agent in the environment gets it and communicates it to the machine. This is the only strategy, to deal with unshared information. Other requirement type called "soft" is vague and imprecise, such as requirements for a system to be "secure," "reliable," or "user-friendly."

Volume : 4 | Issue : 1 | Jan 2014 | ISSN - 2249-555X

3. Conclusion

If the above explained practices are followed to perfection, then requirements engineering, in the real sense, will be complete. We are guaranteed that the specification will be implementable without re-work to any additional information and the requirements would be satisfactory and implementable.

REFERENCE

1. Nancy G. Leveson, Mats Per Erik Heimdahl, Holly Hildreth, and Jon Damon Reese.(2010) Requirements specification for process-control systems. IEEE Transactions on Software Engineering XX(9):684-707. | 2. Michael Jackson and Pamela Zave (2011) Four Dark Corners of Requirement Engineering, Proceedings of the IEEE LXVIII(9):1060-1076,. | 3. Stanley B. Zdonik and David Maier, eds. (2010) Systems, pages 37-46. Morgan Kaufmann. | 4. Lamport .L. (2010) A simple approach to specifying concurrent systems, Communications of the ACM XXXII(1). | 5. Lehman. M. M. (2010) Programs, life cycles, and laws of software evolution. Proceedings of the IEEE LXVIII(9):1060-1076. | 6. Barbara H. Liskov and Stephen N. Z. (2008). Specification techniques for data abstractions, IEEE Transactions on Software Engineering I (1):7-19. | 7. Setrag N. Khoshafian and George P. Copeland. (2011). Object identity, In Readings in Object-Oriented Database. | 8. David L. P. and Paul C. Clements (2006). A rational design process: How and why to fake it. IEEE Transactions on Software Engineering XII (2):251-257. | 9. David L. P. and Madey. J. (2005), Functional documentation for computer systems engineering, Science of Computer Programming XXV: 41-61.. | 10. Michael Spivey. J. (2002) Specifying a real-time kernel, IEEE Software VII (5):21-28. | 11. Michael Spivey. J. (2002) The Z Notation: A Reference Manual, Second Edition. Prentice-Hall . | 12. Jeannette M. Wing.(2001) A specifier's introduction to formal methods. IEEE Computer XXIII (9):8-24. | 13. Auyang S (2004) Engineering: an endless frontier. Harvard University Press, Cambridge. | 14. Brooks F (1996) The computer scientist as toolsmith II.Commun ACM 39(3):61–68. | 15. Bush D (2005) Modelling support for early identification of safety requirements: a preliminary investigation. In: Proceedings RHAS'2005 workshop, 13th international IEEE conference on requirements engineering, Paris. | 16. Cross. N (1994) Engineering design methods: strategies for product design, 2nd edn., Wiley Chichester. | 17. Cross. N (2001) Design cognition: results from protocol and other empirical studies of design activity. In: Eastman C, McCracken.