Software Requirements Implementation and Management

Juho Jäälinoja[1] and Markku Oivo[2]
1 VTT Technical Research Centre of Finland
P.O. Box 1100, 90571 Oulu, Finland
Juho.Jaalinoja@vtt.fi
tel. +358-40-8415550
2 University of Oulu
P.O. Box 3000, 90014 University of Oulu, Finland
Markku.Oivo@oulu.fi
tel. +358-8-553 1900

Abstract. Proper elicitation, analysis, and documentation of requirements at the beginning of a software development project are considered important preconditions for successful software development. Furthermore, successful development requires that the requirements be correctly implemented and managed throughout the rest of the development. This latter success criterion is discussed in this paper. The paper presents a taxonomy for requirements implementation and management (RIM) and provides insights into the current state of these practices in the software industry. Based on the taxonomy and the industrial observations, a model for RIM is elaborated. The model describes the essential elements needed for successful implementation and management of requirements. In addition, three critical elements of the model with which the practitioners are currently struggling are discussed. These elements were identified as (1) involvement of software developers in system level requirements allocation, (2) work product verification with reviews, and (3) requirements traceability.

Keywords: Requirements engineering, software requirements, requirements implementation, requirements management

1. INTRODUCTION

Among the key concepts of efficient software development are proper elicitation, analysis, and documentation of requirements at the beginning of a project and correct implementation and management of these requirements in the later stages of the project. In this paper, the latter issue related to the life of software requirements beyond their initial development is discussed.

The role of software requirements beyond their development is important because all the remaining software development activities are based on requirements. In an ideal software development environment, the agreed requirements would simply be elaborated into working software designs, code, and tests. In reality, however, the agreed requirements change, new requirements are introduced and old ones are removed during the software development. This causes problems in maintaining consistency among the work products. Often the problems with inconsistent work products do not appear until the software and system integration begins.

Problems with requirements implementation and management (further referred as RIM) in software development have been reported by Lindstrom [1], for example. His experiences implied that the development projects typically had drifted away from the original requirements during designing, coding, and unit testing so that at the end the software code did not reflect the changed requirements, added requirements, or eliminated requirements. Problems with changing and incorrect requirements during software development are also reported in [2 and 3]. Possible solutions to inconsistency problems between requirements and resulting work products have been discussed in [4-7].

Unfortunately, the above-mentioned studies offer only domain specific solutions to requirements problems in software development. In this paper, the RIM concept is discussed from a more comprehensive perspective. The main goals of this study are to provide industrial insights into RIM practices, to define a model for RIM that can be used for software process analysis and improvement efforts, and to reveal the most important areas of RIM where further research is needed. The study goals are achieved by first introducing a taxonomy for RIM in Chapter 2 as a basis for the study. Then, in Chapter 3, the taxonomy is used as a framework for case studies to investigate the actual RIM practices in three software organisations. In Chapter 4, a model and further research areas of RIM are presented based on the taxonomy and analysis of the case studies. Finally, in Chapter 5 the most important findings of the study are concluded.

2. TAXONOMY FOR RIM

In general, RIM consists of occurrences to the requirements after they have been initially developed. These occurrences can be categorised into requirements implementation, work product verification, feedback to requirements development, and requirements management subsets. The following sections look into these subsets and their practical implementations in more detail, thereby defining the taxonomy for RIM.

2.1. Requirements Implementation

Requirements implementation is the actual work of transforming requirements into software architectural designs, detailed designs, code, and test cases. The implementation of these work products is less problematic if requirements are unambiguous and quantitative where possible. The work products from architectural designing and software testing are closest to requirements. Generating a set of functional and non-functional requirements into an architecture and valid test cases, however, is not a straightforward task. There is hardly any guidance available on how different architectural decisions affect requirements [8]. Furthermore, it is difficult to develop test cases that appropriately reflect requirements even with reasonable project resources [9].

An automatic approach for transforming requirements into designs, code, and test cases is an appealing concept. Researchers have especially suggested automatic solutions for creating test cases from requirements [10-12]. These approaches often require using formal notations such as Software Cost Reduction (SCR) requirements and therefore are only applicable in specific problem domains. Manual transformation of natural language requirements into work products is, in fact, still the most predominant way of implementing requirements.

2.2. Work Product Verification

Inconsistency problems between requirements and other work products are typical in software development. Software development is conducted by humans and therefore errors may be introduced in any development activity by the developers. However, it is possible to ensure the fulfilment of desired requirements only by providing software work products that are correct and consistent with requirements.

Work product verification can be carried out at different levels of software development. For example, the fulfilment of quality requirements in software architectures can be evaluated with methods such as SAAM [13] and ATAM [14]. The consistency between the requirements and the design and implementation of work products may be verified through requirements tracing and reviews [15-17].

2.3 Feedback to Requirements Development

Requirements development at the beginning of a software project is very rarely completed before the actual software implementation begins. Instead, requirements tend to mature during the development when all the technical restrictions have to be dealt with. Successful requirements development requires a constant flow of information from the actual developers of the software to the requirements analysts.

Feedback to requirements development can be given even before the implementation of software is initiated. Experienced software developers know how the system should be built to support, for example, maintenance and reuse aspects. In addition, developing test cases in parallel with requirements may reveal deficiencies in the intended requirements. Software designers and testers should be involved at least in requirements verification activities to ensure that the requirements can be implemented and are testable. Prototyping the initial requirements before the actual implementation has also long been seen as an excellent method for validating developed requirements.

2.4 Requirements Management

A generally acknowledged fact and problem concerning requirements is their tendency to change. Typically, requirements change due to changes in the system environment or because of the stakeholders gaining a better understanding of their needs as the system is maturing. To cope with this reality, requirements management is needed throughout the development to handle changes to requirements and other work products. Requirements management can be regarded as a process involving a set of techniques for identifying and storing requirements, managing changes, and tracing requirements to other requirements and other software work products.

In practice, requirements management is carried out with a variety of techniques. The storing of requirements can be handled with word processors and databases. Change processes and change analysis techniques are needed for implementing the actual change. Tracing requirements back to higher-level requirements and forward to other work products is needed for analysing changes and can be implemented by means of matrices or specific tracing tools, for example. The integration of the different requirement management techniques has been under the attention of several tool vendors in recent years. Among the popular requirements management tools with integrated features are DOORS and RequisitePro.

3. THE REALITY OF RIM PRACTICES

The previous chapter provided a brief overview of the different elements related to RIM. In theory, all these elements should be properly considered for software development to succeed. To study how theory differs from actual practice, we carried out case studies in three different software organisations developing embedded and application software. One of the goals in these case studies was to investigate what kinds of RIM practices were used and how they were used, based on the taxonomy presented in Chapter 2. Another goal was to determine what the most important and the most problematic practices were. The following table summarises the studied organisations with related background information.

Table 1. Background information of the case organisations

Case organisation	Software developers	Domain	Software process maturity
A	700	Application and embedded software	Not assessed. Pursuing CMM 2
B	350	Embedded software	CMM 3 and ISO 9000
C	25	Application and embedded software	Not assessed. Process improvement based on ISO 15504

Qualitative data for the current state analysis of RIM practices in the three case organisations was collected from multiple sources. First, the current RIM practices and their relevance in the organisations were surveyed among research and development personnel with an open-ended questionnaire. The questionnaire was drawn up based on the taxonomy given in Chapter 2. Then, the most important and problematic issues brought up by the respondents in the survey were further examined with more detailed questionnaires, interviews, and by investigating the software related documentation in the organisations. The following sections are based on the four subsets of the RIM taxonomy and present the most significant findings of the case studies.

3.1. Requirements Implementation Practices

In all of the studied organisations, software requirements were specified mostly in natural language. This indicates that systematic implementation of requirements directly from specific requirements notations to designs, code, and test cases is still not the reality. However, specific notations were sometimes used for handling, for example, real-time specifications in organisation A and definitions for interfaces in organisation B. In organisation C, some diagrams were used for documenting requirements, but these were only used for communication.

The actual work of implementing functional requirements such as designs, code, and test cases was not considered especially problematic. In organisations B and C, requirements documentation included explicit quantitative implementation criteria for certain quality requirements, which helped the developers to get through the implementation. Senior developers in organisation C had the responsibility to ensure the absorption of quality requirements such as portability into the code. Maintainability of software was achieved by following organisation-specific coding standards in all organisations.

3.2. Work Product Verification Practices

The studied organisations employed various techniques in verifying work products against requirements. In organisation B, the correct alignment of software requirements with system requirements was well managed with formal inspections. In contrast, organisation A reported problems with maintaining consistency among the different requirement levels. Problems seemed to arise because of inadequate co-operation among software, hardware, and system disciplines. For example, informal checking of software requirements consistency with system requirements proved to cause problems later in system integration and testing because of the fact that software requirements and those of the other disciplines were not consistent after all.

Consistency checking between software requirements and resulting software work products was conducted with reviews in all organisations. In organisation C, however, these reviews tended to follow a more informal than a formal process. Limited resources simply do not allow extensive checking of consistency between requirements and other work products.

The evaluation of software architecture against requirements was considered important in all three organisations. However, no specific methods for this kind of assessment were used. Only organisation B reported to informally perform architecture trade-off analysis in which quality requirements were covered. A typical situation was reported in organisation C, where architecture design was considered to be well performed, while it was mainly based on designers' expertise.

Due to the great difference in the level of abstraction between requirements and code, no explicit verification of code against requirements was performed regularly. Organisation C reviewed certain parts of the code in which requirements implementation needed to be evaluated. Organisation A verified consistency between code and specific requirements with structured walkthroughs. A static code analyser was used in organisation B to verify the fulfilment of certain quality requirements in code.

3.3. Feedback to Requirements Development Practices

The involvement of software developers in software requirements development was considered important and was reasonably well handled in all organisations. For example, the software designers in organisation B were participating in requirements development right from the beginning. Their job was either to write parts of the requirements or to attend requirements inspections. Besides software designers, also quality assurance personnel were involved in these inspections.

Although the feedback from developers to requirements development was well managed at software level, problems were confronted at system requirements level. Especially in organisation A, decisions concerning the allocation of requirements to software and hardware implementations were mostly made without software adequate expertise. These decisions had sometimes been clearly wrong and expensive to implement.

The involvement of software testers in software requirements development was not well handled in organisations A and B. In organisation B, the development of test cases was started after the architectural design was ready. Because of this, the software testers were often not involved in requirements development at all. In organisation A, the formal process stated that test cases should be written in parallel with requirements. In practice, however, test cases were written after code.

3.4. Requirements Management Practices

Storing requirements was considered an important practice in all case organisations. The organisations typically had enumerable requirements in written requirements specifications for system and software levels. Organisation C used a requirements management tool that included a database for requirements. Organisation A was also planning to introduce a tool for managing requirements.

Requirements change management was not considered especially problematic in organisations A and C. Although these two organisations reported changing requirements as quite a common phenomenon, they had established such change management procedures that the changes did not have any negative effect on software development. In contrast, organisation B considered having problems with changing requirements.

Traceability practices were deficient in all organisations. Tracing from software requirements to system requirements, among software requirements, and from software requirements to following work products was mostly not performed at all because it was considered too complicated and expensive. Requirements tracing from software requirements to system requirements was considered very important or even a critical issue in all organisations. However, the differences in requirements management practices between software and system levels was found to make tracing between them difficult.

Tracing requirements to software designs was also regarded as very important, but it was not formally practiced either. In organisation B, however, requirements were informally traced to software architectures so as to assist in architecture evaluation. Organisation A had a common practice to use matrices in test specifications to link test cases with requirements. In organisation C, it was not always clear how to trace generic requirements to more specific test cases.

An attempt to use a commercial requirements traceability tool was made in organisation B, but it failed because the developers did no have enough experience with the tool and there was no time to train them to use the tool efficiently. Another reason for abandoning the tool was that it could only handle the most trivial relations between requirements, designs, and test documents.

3.5. Summary of Studied Practices

A summary of the studied RIM practices in the case organisations is given in Table 2. The plus and minus symbols in the table indicate whether the organisations supported or neglected a practice. The study implied that

organisation A supported significantly fewer practices than organisation B, which appeared more mature, and organisation C, which was a lot smaller.

Table 2. Summary of the studied RIM practices

Practice	**Case A**	**Case B**	**Case C**
Requirements documentation	+	+	+
Requirements attributes	-	+	+
Measurable quality requirements	-	+	+
Requirements verification	-	+	+
Design trade-off analysis	-	+	-
Design verification against requirements	-	+	-
Code verification against requirements	+	+	-
Software designers involvement in requirements development	-	+	+
Software testers involvement in requirements development	-	-	+
Requirements storage	-	-	+
Change management	+	-	+
Traceability	-	-	-

4. MODEL FOR RIM

The case studies showed that some RIM practices were reasonably well handled in the organisations while others were clearly problematic. Furthermore, the evaluation of the importance of these practices to successful software development also varied in the three cases. Using a cross-case comparison [18], the most important elements that were either performed well or considered problematic in the organisations could be elaborated into a holistic model. The model for RIM is shown in Figure 1. It can be used in software developing organisations as basis for analysing the current state of RIM practices and as a means to improve the software process. Since the RIM practices require a lot of resources, they are most likely to be beneficial in large-scale software development. For smaller projects with constantly changing requirements and very strict time-to-market requirements, the more agile development methods [19] may be more suitable.

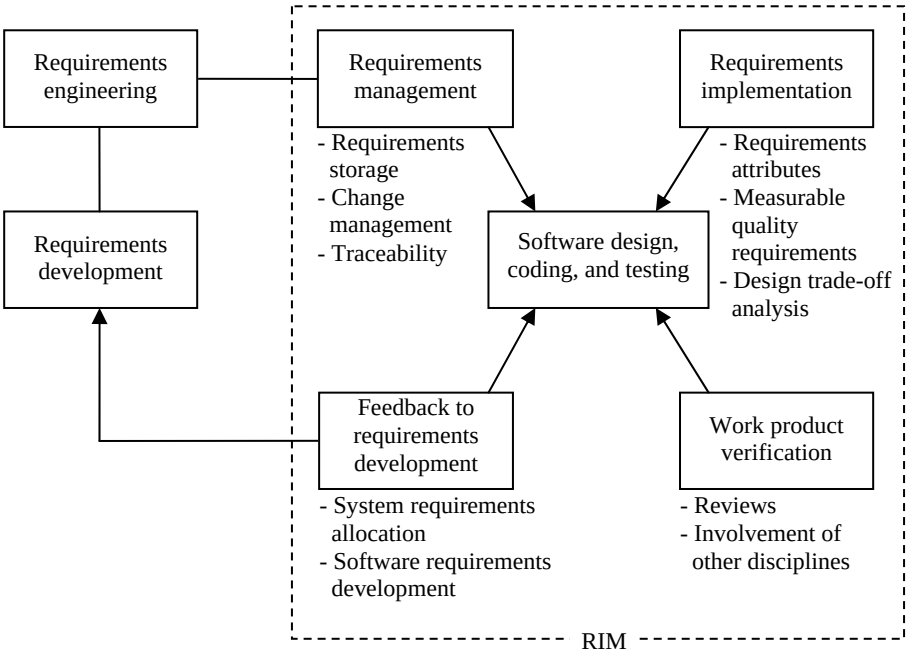

Fig. 1. Model for RIM

For the requirements implementation subset, the essential RIM elements include requirements attributes, measurable quality requirements, and design trade-off analysis against requirements. Work product verification

is effective if reviews are regularly practiced and also if other than software discipline people are also involved in verification activities. The important elements concerning the feedback to requirements development include the participation of software developers in system requirements allocation and in software requirements development. Finally, the essential elements for requirements management are requirements storage, change management, and requirements traceability. The three elements of the model that were identified especially important and problematic in the case studies are discussed more profoundly in the following sections.

4.1. Involvement of Software Developers in System Requirements Allocation

The results from the case studies implied that software developers were reasonably well giving feedback and participating in software requirements development. However, there were serious deficiencies with their involvement in system requirements development. During the system requirements development the allocation of requirements to different disciplines was too often done without adequate software expertise.

In order to improve requirements implementation at software level, researchers and practitioners should pay more attention to improving system requirements allocation from the software point of view. The involvement of software developers during system requirements development may save the project from expensive changes in later development phases. Techniques such as software measurement and prototyping may be used to evaluate development costs and rationalise system requirements allocation procedures.

4.2. Review Practices in Work Product Verification

In view of the fact that automatic transformation of requirements specifications into designs, code, and test cases is not realistic with today's technology, human work dominates the implementation work. This, unfortunately, often leads to emergence of work products that are not fully consistent with requirements. One good solution is to implement solid review practices throughout the development, where humans evaluate human made work products against requirements.

The studied organisations showed reasonably well established review practices where the consistency with requirements was at least informally confirmed. However, review techniques that would explicitly ensure the consistency were not used. This can be ascribed to the fact that such methods are largely missing. One step towards such methods has been taken by the researches at the University of Maryland with their design inspection techniques [15, 16]. A recent industrial experiment with these techniques has shown promising results [20]. Still, more research efforts are needed to create similar techniques for different work products and for different software development domains.

4.3. Requirements Traceability

One of the most important elements of successful RIM is, obviously, tracing software requirements backwards to system requirements and forwards to other software work products. Tracing has long been praised as a vehicle for successful software development. Traceability is sometimes explicitly required in software contracts while it is also encouraged in several well-known software process improvement frameworks such as CMMI and ISO 15504.

However, the reality of requirements tracing is currently far from the ideal situation. The case studies showed that such problems with traceability as insufficient resources and imbalance of return on investment recognised a decade ago [21] still exist today. Nevertheless, a positive finding from the case studies was that requirements tracing was regarded as an important practice by the organisations. The problem is that they do not have the means to do it effectively.

5. CONCLUSIONS

The life of requirements beyond their initial development has until now been studied only from relatively narrow perspectives. This study contributes to the knowledge of software engineering by providing industrial insights into this issue from the domain of embedded software. The RIM practices employed in the industry were observed through case studies in three software organisations. The case studies showed that some of the practices such as requirements documentation were well managed whereas other important practices such as requirements traceability were difficult to implement.

Based on the observations from the industry, a model for RIM was defined containing the essential elements needed for the implementation and management of requirements. Due to the limited number of case studies considered in this research, no general conclusions should be drawn about the validity of the model as a reflection of the general state in software industry. However, the developed model can be used as a starting point for software process analysis and improvement efforts in software organisations and as a guideline for further research efforts.

In addition to the model, three especially important areas of RIM were presented based on the analysis of the case studies. These areas were identified as (1) involvement of software developers in system level requirements allocation, (2) work product verification with reviews, and (3) requirements traceability. More research is needed to identify exactly what kinds of problems practitioners are facing in these areas and how these problems can be solved.

6. REFERENCES

[1] Lindstrom, D.R., "Five Ways to Destroy a Development Project", IEEE Software, 1993, Vol. 10, no. 5, pp. 55-58.

[2] Harker, S.D.P., Eason, K.D., and Dobson, J.E., "The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering", Proc. IEEE International Symposium on Requirements Engineering, San Diego, USA, 1993, pp. 266-272.

[3] Komi-Sirvio, S. and Tihinen, M., "Great Challenges and Opportunities of Distributed Software Development - An Industrial Survey", Proc. 15th International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, USA, 2003.

[4] Kozlenkov, A. and Zisman, A., "Are their Design Specifications Consistent with our Requirements?" Proc. IEEE Joint International Conference on Requirements Engineering, Essen, Germany, 2002, pp. 145-154.

[5] Chechik, M. and Gannon, J., "Automatic Analysis of Consistency between Implementations and Requirements: a Case Study", Proc. Tenth Annual Conference on Computer Assurance 1995, pp. 123-131.

[6] Chechik, M. and Gannon, J., "Automatic Analysis of Consistency between Requirements and Design", IEEE Transactions on Software Engineering, 2001, Vol. 27, no. 7, pp. 651-672.

[7] Olsson, T. and Grundy, J.C., "Supporting Traceability and Inconsistency Management Between Software Artifacts", Proc. IASTED International Conference on Software Engineering and Applications, Boston, USA, 2002.

[8] Grünbacher, P., Egyed, A., and Medvidovic, N., "Reconciling Software Requirements and Architectures: The CBSP Approach", Proc. Fifth IEEE International Symposium on Requirements Engineering 2001, pp. 202-221.

[9] Weber, M. and Weisbrod, J., "Requirements Engineering in Automotive Development - Experiences and Challenges", Proc. IEEE Joint International Conference on Requirements Engineering, Essen, Germany, 2002, pp. 331-340.

[10] Tahat, L.H., Vaysburg, B., Korel, B., and Bader, A.J., "Requirement-Based Automated Black-Box Test Generation", Proc. 25th Annual International Computer Software and Applications Conference, Chicago, USA, 2001, pp. 489-495.

[11] Gupta, P., Cunning, S.J., and Rozenblit, J.W., "Synthesis of High-Level Requirements Models for Automatic Test Generation", Proc. Eighth Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, Washington, USA, 2001, pp. 76-82.

[12] Gargantini, A. and Heitmeyer, C., "Using Model Checking to Generate Tests from Requirements Specifications", Proc. ESEC/FSE'99, Toulose, France, 1999.

[13] Kazman, R., Abowd, G., Bass, L., and Clements, P., "Scenario-Based Analysis of Software Architecture", IEEE Software, 1996, Vol. 13, no. 6, pp. 47-55.

[14] Kazman, R., Klein, M., Barbacci, M., Lipson, H., Longstaff, T., and Carrière, S.J., "The Architecture Tradeoff Analysis Method", Proc. Fourth IEEE International Conference on Engineering of Complex Computer Systems, Monterey, USA, 1998, pp. 68-78.

[15] Travassos, G.H., Shull, F., Fredericks, M., and Basili, V.R., "Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality", Proc. OOPSLA '99, Denver, USA, 1999, pp. 47-56.

[16] Travassos, G., Shull, F., Carver, J., and Basili, V., Reading Techniques for OO Design Inspections. Technical Report CS-TR-4353, UMIACS-TR-2002-33. 2002, University of Maryland.

[17] Knight, J.C. and Myers, E.A., "An Improved Inspection Technique", Communications of the ACM, 1993, Vol. 36, no. 11, pp. 51-61.
[18] Eisenhardt, K.M., "Building Theories from Case Study Research", Academy of Management. The Academy of Management Review, 1989, Vol. 14, no. 4, pp. 532-550.
[19] Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J., ”Agile software development methods. Review and analysis”, 2002. VTT Electronic, Espoo. VTT Publications : 478.
[20] Conradi, R., Mohagheghi, M., Arif, T., Hegde, L.C., Bunde, G.A., and Pedersen, A., "Object-Oriented Reading Techniques for Inspection of UML Models -- An Industrial Experiment", Proc. European Conf. on Object-Oriented Programming ECOOP'03 2003: Springer, LNCS 2743.
[21] Gotel, O.C.Z. and Finkelstein, C.W., "An Analysis of the Requirements Traceability Problem", Proc. First International Conference on Requirements Engineering 1994, pp. 94-01.