PREREQUISITES FOR EFFECTIVE REQUIREMENTS MANAGEMENT

Konrad Grzanek

IT Institute, Academy of Management, Łódź, Poland
kgrzanek@swspiz.pl, kongra@gmail.com

Abstract

Despite an undeniable progress in the whole software creation process, software development is still more art than science. The requirements analysis is a highly critical step in the software life-cycle. Requirement managements errors are the most common errors in the software projects. The proper and effective requirements management saves the overall project costs. The key motivation behind this work was opening a way of finding approaches to managing the requirements appearing in such large software projects as compilers for various programming languages. This paper is an introduction to a full presentation of requirements management solution in which the requirements and implementation information is placed directly in the source code. We concentrate on describing a context in which the requirements management process takes place, trying to present the most interesting existing solutions, indicating the problems and opening a discussion on what ways to follow in the future scientific research.

Key words: requirements engineering, requirements abstraction, functional programming

1 The State of the Art in Requirements Management

According to the commercial surveys conduced back in the ’90s an average US software project overran its budgeted time by 190%, it’s budgeted costs by 222%, and delivered only 60% of the planned functionality. Only 16% of projects were delivered at the estimated time and cost, and 31% of projects were canceled before delivery, with larger companies performing much worse than smaller ones (source: [1]). Martyn Thomas also mentions:

> „A UK survey, published in the 2001 Annual Review of the British Computer Society showed a similar picture. Of more than 500 development projects, only three met the survey's criteria for success. In 2002, the annual cost of poor quality software to the US economy was estimated at $60B."

Despite an undeniable progress in the whole software creation process, software development is still more art than science (after [3]). Most researchers point out the following causes of software process failures [3]:

- Poor requirements management. We forge ahead with development lacking user input and without a clear understanding of the problem we are attempting to solve.
- Poor change management. Changes to requirements and other development products are inevitable; yet we rarely track them or understand their impact.
- Poor quality control. We have poor measures for system quality, little knowledge of processes that affect quality, and no feedback to modify the process after witnessing the effects of a particular development strategy.
- Little control of schedules and costs. Accurate planning is the exception while unrealistic expectations are the norm.

It is a fact universally acknowledged in the software engineering world that requirements analysis is a highly critical step in the software life-cycle [2]. The lack of the ability to specify, control and manage the software project requirements causes the loss of control over the overall system behavior, it's design and quality [3]. The proper and effective requirements management saves the overall project costs due to the following reasons (as stated in [3]):

- Requirement errors typically cost well over 10 times more to repair than other errors.
- Requirement errors typically comprise over 40% of all errors in a software project.
- Small reductions in the number of requirement errors pay big dividends in avoided rework costs and schedule delays.

Moreover, the requirement managements errors are the most common errors in the software projects. No wonder the issue is seen as one of the fundamental issues in the field both by scientific researches as well as organizations like Software Engineering Institute (SEI) with their Capability Maturity Model. In CMM the requirements management is one of the first steps to achieving process maturity and the key area that must be addressed to move from Level 1 to Level 2 [3].

We define a requirement as a capability or feature needed by a user to solve a problem or achieve an objective. There are two major kinds of re-

quirements, the functional and the non-functional ones (abbreviated NFR). Some management approaches try to treat these two categories uniformly, but often they are treated separately due to their apparent differences in nature (e. g. [4]) - the functional requirements specify each function that a system must be capable of performing, whereas the NFRs specify how the system is going to be implemented to achieve it's goals and what it's quality attributes will be.

The requirements engineering as defined in [5] and [6] is:

> „the branch of software engineering concerned with the real-world goals for, functions of, and constraints on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families."

This discipline inevitably breaks the borders of multiple system views, because – as stated in [6] - software cannot function in isolation from the environment in which it operates and in which it is embedded. So in fact the requirements engineering may be treated as a branch of systems engineering. The paper [6] also states that requirements may and should undergo formal treatment, i. e. the formal description and reasoning. Present article addresses the problem of the formal describing and managing the requirements in a persistent way.

Works by Zave and Jackson expose similar conclusions related to the multidisciplinary character of requirements engineering. In [7] the authors put a particular emphasis on the impact of the environment onto the requirements engineering process. They state that the descriptions of the requirements should be in fact the environment's descriptions. Another problem they try to address are the implementation bias while defining the requirements (especially in the early stages/high abstraction layers) and the role of knowledge management in the whole process. Some design and implementation achievements presented in the present paper loosely refer to the knowledge management area.

Hofmann and Lehner [8] underline the unquestionable importance of possessing a deep domain knowledge on increasing the probability of software project success. Consequently the requirements engineering is the key factor here, together with the experts' knowledge as well as the stakeholders' competence. According to this the requirements engineering is a multidisciplinary and highly competence-demanding field.

This multidisciplinary character refers also to the possible applications for the requirements management. The discipline is by no means limited to the software domain. The paper [9] describes a requirements management framework that enables health information custodians (HIC) to document and track compliance with privacy legislation as the legislation and hospital business

processes evolve. An interesting graphical notation called the User Requirements Notation (URN) is given there together with it's major complementary notations, namely the Goal-oriented Requirements Language and Use Case Maps.

Requirements representation is very important with respect to the potential algorithmic processing. For example [11] gives a broad and systematic review of existing literature works that transform textually represented requirements into analysis model. The major reason is rooted in model transformation being one of the basic principles of Model Driven Architecture. According to the paper building a software system consists of a sequence of transformations, starting from requirements and ending with implementation. The problem of textual representations and processing of requirements will also be addressed further in our works.

In [10] there are mentioned the relations between requirements and high-level testing methodology called Abstract Testing. The article states that „[...] often a one-to-one correspondence between abstract test cases (resp. verification scenarios) and requirements can be achieved, which links abstract testing much more closely to the requirements and facilitates construction and maintenance of abstract test cases". The interesting feature of this methodology as a whole is an existence of a close relationship of the verification scenarios (and so – indirectly – the requirements) to the source code by the fact of an automatic checking the scenarios against the sources performed by a source model checker.

The requirements modeling with a combination of SysML and UML are described in [12]. The work is of a special significance because it presents the problem in the context of real-time systems specification. A classification of user requirements is also proposed there.

2 The Idea of Source Code as the Requirements Database

Now we should take a look at the existing approaches and tools automating the requirements management. According to [16] most requirements management tools perform the same core functions:

- They allow the system developer to import large documents from a variety of standard word processing formats.
- These documents can be split up into separately managed document elements.
- The document elements are subject to a rigorous change and version control regime.
- Relations can be established between document elements and attributes can be associated with the document elements and often the relations.

- A variety of document views can be generated using both attributes and relations, generally specific traceability views such as traceability matrices.
- Document templates can be set up and used to create new composite documents.
- Scripting or query languages provide support for the retrieval of information and the development of project specific views.
- Simple checks to ensure structural integrity of documents may be performed.

An apparent importance of attribution and labeling shows up. In fact, the labels are the core of our planned approach. The tool we start working on will also possess an effective querying mechanisms.

When referring to the architecture of requirements management tools [16] states that these tools have much in common: „They are generally based on a document repository, which may either be hosted on top of an industry standard database (relational or object-oriented) or a specifically crafted file store [...]. Most tools provide some simple control for multi-party editing of documents, the granularity of this control is dependent upon the underlying repository. At the front end, the tools generally appear similar to standard document processors. From a user interface standpoint, they provide a number of tools to support work with large hierarchical documents including the ability to work seamlessly in different document views."

[16] notices a very important weakness of these tools; they are in general process-free. The obvious yet not fully realized yet solution is „to integrate requirements management tools with a work-flow or process engine. Despite this being an obvious answer it is not very straightforward to achieve." One possible solution would make a step towards integrating programmers' personal information management with requirements management and their basic professional activity: working on the source code of implemented systems.

More features of an ideal requirements management system are presented in [17]. They extend the list of previously mentioned desired features with such elements as:

- Using effective information models
- Supporting various views of the same data
- Handling formal change requests for the requirements
- Keeping the history of requirements change
- Allowing base-lining
- Effective tool integration

and many more.

The key motivation behind this work was finding a way to manage the requirements appearing in such large software projects as compilers for various

programming languages. The Java 6 Language Specification is over 600-pages document containing lots of facts about the Java language run-time and compiler. Our goal is to deal with all those facts in an organized way. There are the following conclusions related to our situation:

1. There are huge amounts of facts in such a system, expected number reaches thousands of facts.
2. The facts are distributed, spread around many chapters of the source document. It closely resembles real-life situations that may appear in software projects of different nature.
3. Some information may be ambiguous and their transformation into the requires an active support from the programmer/designer.
4. The modules of the system described by these facts will be implemented by single developers who must have a clear view of what is to be done.
5. The requirements management system should not only help the programmer to organize these large volumes of information, but should also give him some help during the process of organizing facts.

We decided to use a unique approach of integrating the requirements management with source code. This approach is inspired by a ***homo-iconicity*** of the languages from the Lisp family of programming languages. Our solution is an embedded domain-specific language based on Clojure [18]. This DSL wins the following for the analysts, designers and programmers:

- Editing source code is a primary activity every programmer undertakes on every work-day. Putting the act of reading/writing the requirements into source code increases the comfort of this – sometimes boring – activity.
- It also affects the designers and other people not involved directly in the implementation phase, because it opens an effective channel of communication between – for instance – a system analyst and a coder; the analyst writes a requirement directly in a compilation unit, the programmer reads it and perform further steps to gain the required functionality.
- The presence of requirements in compilation units allows to interweave the them (their definitions formally speaking) with source code snippets being their direct implementations or implementation parts. This point is especially important because an act of locating requirements in pure (not instrumented with requirements or requirement-related tags) source code is a tedious and hard to solve problem. Further works on this can be found in [13, 14][1].
- A compilation unit keeping some requirements may be tracked and managed by a source management and revision control system, such as Git [19]. An immediate consequence is the ability to manage the requirements

1 Very interesting works related to real-time systems programming, source-code verification and requirements management in Ada programming language were presented in [15].

versions, because a requirement change is a change in the compilation unit. All version control system's goodies, including the possible encryption and the overall robustness of a distributed versioning system are there to be used.

3 Summary

The key motivation behind this work was opening a way of finding approaches to managing the requirements appearing in such large software projects as compilers for various programming languages. This paper may be treated as an introduction to a full presentation of requirements management solution in which the requirements and implementation information is placed directly in the source code. We concentrated on describing a context in which the requirements management process takes place, trying to present the most interesting existing solutions, indicating the problems and opening a discussion on what ways to follow in the future scientific research.

The main motivation for the conceptual creation and implementation of an innovative requirements management system is the urge to control the complexity (especially the non-accidental one) of large software projects, such as the implementation of a static analyzer of a formally described programming language or large modeling environments, such as the environments in which some biological structures and behaviors could be modeled (e. g. human immune system). The results of applying the requirements management system in future will be presented in future papers.

References

1. Thomas M., 2003, *The Modest Software Engineer*, Proceedings ISADS 2003, IEEE Press, pp. 169-174
2. Dardenne A., van Lamsweerde A., Fickas S., 1993, *Goal-directed Requirements Acquisition*, Science of Computer Programming, Vol. 20, pp. 3-50
3. Davis A. M., Leffingwell D. A., 1995, *Using Requirements Management to Delivery of Higher Quality Applications*, Rational Software Corporation
4. Ebert Ch., 1997, *Dealing with nonfunctional requirements in large software systems*, Annals of Software Engineering 3 (1997), pp. 367-395
5. Zave P., 1997, *Classification of Research Efforts in Requirements Engineering*, ACM Computing Surveys, 29(4), pp. 315-321
6. Nuseibeh B., Easterbrook S., 2000, *Requirements engineering: a roadmap*, ICSE '00 Proceedings of the Conference on The Future of Software Engineering, pp. 35-46

7. Zave P., Jackson M., *Four dark corners of requirements engineering*,1997, ACM Transactions on Software Engineering and Methodology (TOSEM) Volume 6 Issue 1, Jan. 1997, pp. 1-30
8. Hofmann H.F., Lehner F., 2001, *Requirements Engineering as a Success Factor in Software Projects*, IEEE Software Jul/Aug 2001, pp. 58-66
9. Ghanavati S., Amyot D., Peyton L., 2007, *A Requirements Management Framework for Privacy Compliance*, Proc. of the 10th Workshop on Requirements Engineering (WER'07), pp. 149-159
10. Merz F., Sinz C., Post H., Gorges T., Kropf T., 2010, *Abstract Testing: Connecting Source Code Verification with Requirements*, 2010 Seventh International Conference on the Quality of Information and Communications Technology (QUATIC), pp. 89-96
11. Yue T., Briand L.C., Labiche Y., 2011, *A systematic review of transformation approaches between user requirements and analysis models*, Requirements Engineering Volume 16 Issue 2, June 2011, pp. 75-99
12. Santos Soares M., Vrancken J., Verbraeck A., 2011, *User requirements modeling and analysis of software-intensive systems*, The Journal of Systems and Software 84 (2011), pp. 328-339
13. Eisenbarth T., Koschke R., Simon D., 2003, *Locating Features in Source Code*, IEEE Transactions on Software Engineering, pp. 210-224
14. Eaddy M., Aho A.V., Antoniol G., Gueheneuc Y.G., 2008, *CERBERUS: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis*, ICPC 2008. The 16th IEEE International Conference on Program Comprehension, pp. 53-62
15. Ruiz J.F., Comar C., Moy Y., 2012, *Source Code as the Key Artifact in Requirement-Based Development: The Case of Ada 2012*, Ada-Europe 2012, Stockholm
16. Finkelstein A., Emmerich W., 2000, *The future of requirements management tools*, In: Quirchmayr, G and Wagner, R and Wimmer, M, (eds.) Information Systems in Public Administration and Law. Oesterreichische Computer Gesellschaft (Austrian Computer Society)
17. Hoffmann M., Kuhn N., Weber M., 2004, *Requirements for requirements management tools*, In Proceedings of the IEEE International Requirements Engineering Conference (RE'04), pp. 301-308
18. *The Clojure Language Website*, 2012, http://clojure.org
19. *Git, Website* 2012, http://git-scm.com/