Managing Evolving Requirements in an Outsourcing Context: An Industrial Experience Report

Marco Lormans Hylke van Dijk
Delft University of Technology
The Netherlands
{M.Lormans,H.W.vanDijk}@ewi.tudelft.nl

Arie van Deursen
CWI and Delft University of Technology
The Netherlands
Arie.van.Deursen@cwi.nl

Eric Nöcker Aart de Zeeuw
LogicaCMG, Technical Software Engineering, The Netherlands
{Eric.Nocker,Aart.de.Zeeuw}@logicacmg.com

Abstract

In this paper we discuss several difficulties managing evolving requirements by means of an industrial case study conducted at LogicaCMG. We report on setting up a requirements management system in an outsourcing context and its application in real-life. The experience results in several lessons learned, questions to be answered in the future on how to manage evolving requirements, and solution directions. We propose a conceptual framework of a requirements engineering system tailored for outsourcing environments, which captures the experience results.

1. Introduction

In the IT outsourcing services business it is quite common that the requirements of a new product are provided by an external stakeholder, the client. The client wants a new product and needs you to develop it. In order to eliminate risks of budget overruns, your client may insist on a fixed price agreement. The requirements document often forms the contract. This document typically is the outcome of an elicitation process conducted by the client.

A key problem in outsourcing development, however, is the evolution of requirements: no matter how thorough the requirements specification has been set up, the requirements for any non-trivial system will change, not only after the system has been built, but also during the process of implementing the system. This evolution of requirements can be due to many reasons, including changing business needs or market and technology developments [10]. In addition to that, the process of designing, implementing, and writing test cases for requirements will increase insight in the problem domain, which may well lead to modifications on the initial set of requirements.

A major risk of evolving requirements is that the requirements document itself becomes inconsistent. This may lead to a system that cannot be implemented, misinterpretations or false assumptions by developers, and delivery of a system that will not be accepted by the client.

Requirements management is the requirements engineering activity that aims at controlling changes made to requirements. The purpose of this paper is to analyse the impact of outsourcing on requirements management. To that end, we study what requirements management techniques an IT solution provider can adopt when accepting an outsourcing contract. In particular, we evaluate the various methods and techniques used by LogicaCMG — a major international player in IT services and wireless telecoms — in order to manage evolving requirements of a traffic monitoring system they are implementing for an external customer.

This paper is primarily an experience report. We believe that collecting and organising the experiences obtained by LogicaCMG in a case like this is important for a number of reasons. First, our discussion of requirements management problems as occurring in a state of the art industrial software engineering project may help practitioners in obtaining a better understanding of the problems in their own projects. Second, we discuss which requirements management methods and techniques were actually used, and analyse why these worked well or why they were not satisfactory. This provides an evaluation of existing techniques, which will be valuable to researchers as well as practitioners. Last but not least, we establish a connection between the problems we encountered and the published literature in the software evolution and requirements management literature. From this, we derive a number of research questions in the area of requirements evolution.

The case study discussed in this paper was carried out as part of the MOOSE project [1]. MOOSE is an ITEA project that aims at improving software quality and development productivity for embedded systems, by adapting, tailoring,

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

and combining technologies to fit a specific – industrial – situation.

The remainder of this paper is organised as follows. In Section 2 we provide background information and discuss the key concepts in requirements management. Then, in Section 3, we provide an overview of the context of the case study, discussing the application domain, as well as standards and tooling. In Section 4 we present issues pertaining to IT outsourcing including the requirements for a requirements management system specifically for outsourcing. In Section 5 we zoom in on the LogicaCMG case, and explain how requirements evolution was tackled in this project. We reflect on this case study in Section 6, where we provide a discussion of the observations and lessons learned. The discussion ends with a proposal for a conceptual framework of a requirements engineering system in the context of outsourcing, which captures our observations. We conclude this paper with a summary of the key contributions and directions for future research.

2. Requirements Management

Requirements engineering is often split into two main areas of activities: requirements specification and requirements management [11]. In our definition, requirements specification concerns activities related to elicitation, analysis, documentation, and validation of requirements. It primarily deals with the content of the requirements.

Requirements Management concerns activities related to controlling and tracking of changes to agreed requirements, relationships between requirements, and dependencies between the requirements specifications (documents) and other specifications produced during the systems and software engineering process [8]. We purposefully do not adopt the definition of Leffingwell and Widrig [9], who include elicitation, organisation, and documentation of requirements under management activities. In our setting, requirements management is primarily a supportive process, which helps to manage evolving requirements throughout the system's lifecycle.

2.1. Requirements Management Systems

Because of the growing number and volatility of requirements, requirements management systems (RMS) have been developed. In [13] various reasons for using a RMS are discussed. In the context of outsourcing, status tracking, and effective communication and interaction with stakeholders are important.

In [3] the importance of communication in the requirements management process is emphasised. According to Al-Rawas and Easterbrook consistency checking of newly introduced requirements often implies re-establishing communication with the stakeholder of the existing requirements. Traceability is of utmost importance for re-establishing this

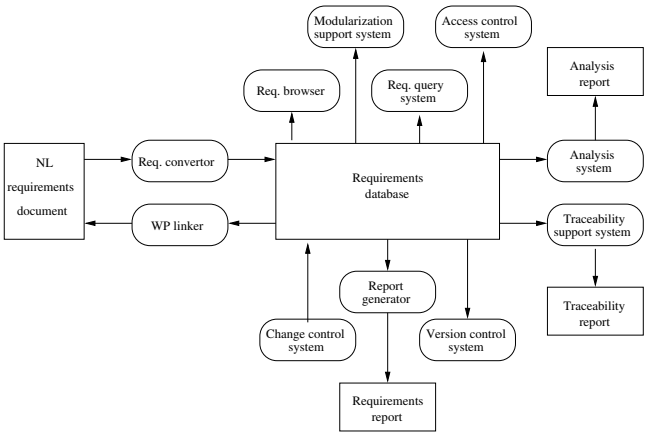

Figure 1. A Requirements Management System

communication in evolving requirements. The inability to trace these stakeholders and their related information is the crux of the requirements traceability problem [6]. In practice keeping the traceability links consistent is a serious challenge [7].

2.2. Features of a RMS

A typical RMS stores its requirements repository and provides a number of features to support requirement management activities related to maintenance, evolution, traceability, and change management. Figure 1 depicts the typical features of a RMS, based on a diagram of a RMS discussed in [8]. We added features from a discussion in [13]. This results in the following list of features for any RMS:

- a browser; to support navigation in the set of requirements, e.g. view requirements subsets,
- a query system; to support retrieval of specific requirements from the set of requirements or related requirements,
- a traceability support system; to support management of links to other system elements and the generation of traceability information,
- a report generator; to support generation of all kinds of different reports related to requirements,
- an interface to external documentation; a typical implementation includes a requirements converter and a word processor (WP) linker, to support conversion of the natural language representation (NL) of the requirements to a database format and back again,
- a change control system; to support management of change requests and links to affected requirements,

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

- a version control system; to support management of different versions of single requirements,
- an analysis system; to perform all kind of analysis on the set of requirements, e.g. impact analysis, status tracking, but also if a requirement is an orphan or not,
- an access control system; to control the access rights of users. Not all users are allowed to browse or edit the complete set of requirements,
- a modularisation support system; to support grouping of requirements, e.g. related to a specific quality attribute or piece of functionality.

The list is not exhaustive. The environment and situation determine the features that should be implemented and at what level of detail. For example, in many situations it is not required to implement an access control system.

3. Case Study Context

This section introduces the application domain of our case study: a traffic monitoring system (TMS). Besides the TMS we will also introduce MIL-std 498 and the applied tooling at the outsource vendor's site.

3.1. Traffic Monitoring System

Our case study involves a traffic monitoring system (TMS [1]), which is an important part of a traffic control and logistics system for a densely populated traffic system. The main purpose of TMS is to record the positions of vehicles on the net. These recordings are used to adjust the schedules of running and planned vehicles as well as operating the necessary signalling. The system is business critical, but not safety critical.

TMS retrieves its data from multiple measurement sources. Its functionality includes distribution of information, occupation management, and track management. Amongst others, the TMS informs client systems with real-time, consistent, and unambiguous data about vehicle positions. It maintains vehicle movement information (identification and order) at the borders of unmonitored areas and maps this information to actual vehicle movements.

The TMS requirements have been set up by the TMS owner, the design and implementation is done by LogicaCMG. The design makes use of various UML models, and includes correctness proofs for critical state diagrams. The implementation is being written in C++.

[1] Details of the case have been modified and made anonymous in order to protect the interests of the customer. We believe that these changes do not materially affect the experiences and results discussed in the paper.

3.2. Documentation structure and Tooling

The interaction between the outsourcer, the owner of the TMS, and the outsourcing vendor, LogicaCMG, is organised around MIL-std 498 documentation standard. This standard was developed at the United States Department of Defence to realize a common software development standard [5]. Only part of the standard is implemented in our case study (see Section 5.1).

The outsource vendor, in our case study, chose to use Rational™ tooling to support the requirement management activities as well as parts of its development processes. First of all Rational RequisitePro is applied for managing the requirements. It maintains a repository of requirements: their identification, description, and relations, augmented with other information such as their design rationale, and test criteria. The data is stored in an external database, in our case a Microsoft Access database. The tool has a close relation with Microsoft Word to capture and edit the requirements. Rational Rose is used to develop various UML diagrams. These diagrams primarily describe the system design, but are also used to explain the details of a requirement. Unlike textual requirements, the diagrams are not managed by Rational RequisitePro and therefore ignored, due to a technology discrepancy. Rational SoDA is applied for generating reports according to a prescribed template. The necessary information is taken from Rose and RequisitePro. Finally configuration management, including version and change control, is implemented using Telelogic Synergy, which manages all documents and repositories.

4. Outsourcing

In Section 2 we discussed requirements management in general, and in the previous section we provided the necessary background material (on traffic monitoring, the MIL-std 498, and tool support) required to analyse the case at hand. In this section, we analyse the implications of adopting explicit, tool-supported requirements management in an outsourcing context. We are not aware of other papers discussing these implications, although some material can be found in [4, 12].

Outsourcing of system development, integration, and maintenance is an important trend in IT services [2]. For a client, the outsourcer, contracting out work is a cost-effective way to hire expertise at a fixed price and get in return a system (or service) with a predefined quality, which includes timely delivery. The observable characteristics of the system, the requirements, are an essential part of the contract between the client and the outsource vendor. Requirements are accompanied by acceptance tests providing means to validate their correct implementation of the requirements.

The details and structure of the specification of the requirements have significant influence on the way of work-

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

ing for the outsource vendor. The short-term concern for the vendor is to establish a cost-effective way to comply with the obligations of the contract; covering the set of requirements and passing the acceptance tests. The vendor, however, also has long-term concerns, such as winning an extended contract for maintaining the developed system and winning similar contracts with other customers.

A number of issues have to be resolved for successful outsourcing of systems, which in [2] are summarised as: control, ownership, development paradigms, assurance, and system decomposition. Typically these issues are addressed in the contracts and agreements.

Control involves concerns like quality, security, and confidentiality. It also handles the responsibility of integration. Ownership is a complicating factor when changes to requirements or implementations are required. Development paradigms are articulated for the sake of compatibility and communication, assurance defines acceptance tests. The responsibility of system decomposition has to be clearly communicated because changes occur in every outsourcing contract. Changing client requirements, clarifications, or design trade offs may induce changes that need a modification to the chosen decomposition.

4.1. Requirements Engineering Process

In an outsourcing context the responsibilities in the requirements engineering process are distributed over the outsourcer and the outsource vendor. Figure 2 outlines the process and identifies possible hazards using the MIL-std 498.

The outsourcer executes the elicitation process and is responsible for the documentation of the requirements in a System Requirements Specification (SRS). This SRS is the basis of a negotiated contract between the outsourcer and the outsource vendor. Given the SRS, the actual development of the product is then done by the outsource vendor and documented in a System Design Description (SDD).

During development of the product the outsource vendor can run into some conflicting or ambiguous requirements. These are noted as issues and should be renegotiated with the outsourcer. Concurrently, the outsourcer developes new ideas that should also be implemented in the system (SRS′). The synchronisation of these parallel activities are a potential hazard. There are two basic resolutions, either the outsource vendor holds back the SDD and issues, or the outsourcer holds back the SRS′. In the first case the outsource vendor incorporates the SRS′ into the SDD, forming an updated version of the SDD′. In the second case the outsourcer resolves the issues of SDD before releasing the updated version of the SRS. This latter case is illustrated in the second iteration of Figure 2. A hybrid version for this synchronisation process, although possible, yields a difficult process for keeping all the system artifacts consistent. Note that the evolution has two sources: advancing insights from the outsourcer and advancing insights from the outsource vendor.

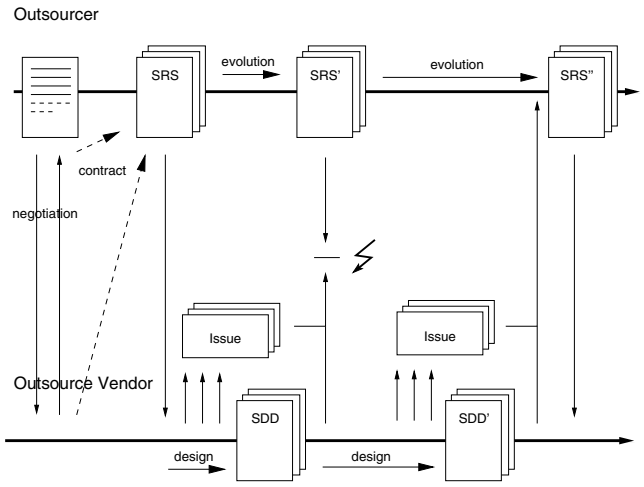

Figure 2. RE Process in Outsourcing Context

4.2. Requirements Management Tool Implications

In the ideal case, the input to the RMS is a set of *frozen* TMS requirements, laid down in the contract with the client. With these correct, complete, and non-conflicting requirements, the system is implemented and simply passes the final system acceptance test.

Passing this test ends the project.

However, practice is less cooperating. Generally, the system requirements are incomplete and ambiguous. This has far reaching consequences: the RMS must support *evolution*. During the design process any ambiguity has to be resolved. This may yield an update, insertion, or deletion of an identified requirement. In other words the RMS must support a process that can handle changes effectively, including maintaining a consistent set of requirements.

With respect to the general features of a requirements management system, a RMS in the context of outsourcing must pay special attention to:

- Change Management; project members have to be aware of the up-to-date baseline of requirements, evolutions, and anticipated changes.
- Quality assurance; the coherence and applied terminology of the requirements must be verified before delivering a design. Conflicting requirements need to be resolved in interaction with the outsourcer.
- Issue Tracking; during the project questions and change requests are communicated (possibly over multiple channels). Their status and history need to be recorded and accessible.

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

- Test reporting; tests are induced by the requirements. The outsource vendor should be able to show via reporting if and how the requirements are fulfilled.
- Status reporting; status and other attributes of requirements should be translated to numbers and should be used in status reporting.
- Flexible modularisation; the system decomposition of the outsourcer does not necessarily comply with the preferred decomposition of the system by the outsource vendor. With flexible modularisation support the vendor can choose to internally use a different modularisation than used externally in the communication with the outsourcer.

The above features concern communication and interaction with stakeholders and is primarily the result of interviews with members of the development team of LogicaCMG.

5. Case Study Implementation

This section describes the requirements management process as implemented at LogicaCMG for the TMS case. Two important, previously introduced, concepts of this RMS are: the document structure MIL-std 498 and the tooling by Rational. In this section we will discuss how LogicaCMG applies these concepts. We first introduce the applicable parts of MIL-std 498, then describe the traceability model that helps realising the requirements for the RMS of the outsource vendor, after that the tooling is discussed, and some case statistics presented.

5.1. Document Structure

The prime responsibility of the client is to provide the requirements, the prime responsibility of the vendor is to implement these requirements. Only part of the MIL-std 498 is implemented to provide the requirements in our case study. The essentials are captured in Figure 3.

The client provides, the Operational Concept Description (OCD), System/Subsystem Specification (SSS), System/Subsystem Design Description (SSDD), and the System Requirements Specification (SRS) (see Figure 3). The client also provides the corresponding acceptance tests at super-system level including the System Test Plan (STP). All documents are plain-text Microsoft-Word documents.

The documents are the input for the development team of LogicaCMG. They split the documents into smaller units, and add design information. The SRS is used for creating a System Design Description (SDD) and System Test Descriptions (STD). In fact the client and LogicaCMG used the SRS, SDD, and STD as system specifications rather than software specifications as is usual in the MIL-std 498.

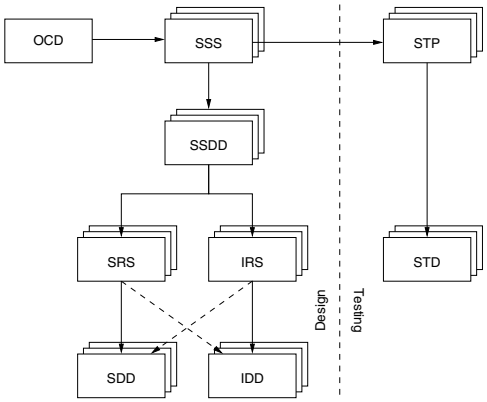

Figure 3. MIL-std 498 outline

5.2. Requirements Traceability Model

The traceability model consists of a number of different requirement types. We identified a number of them, which we believe are typical for the outsourcing business. The requirement types are: system requirement, issue, design decision, assumption, and non conformance.

A *system requirement* is a traditional requirement type that describes *what* the system should be able to do, but not *how* the system will do it. LogicaCMG categorised its system requirements into functional, non-functional, and design constraints and are part of the SRS's, and IRS's.

An *issue* is a type of requirement introduced by LogicaCMG that marks a point of attention that needs further investigation and possibly negotiation with the outsourcer. Issues are the communicating vehicle with the client and other stakeholders. An issue often concerns multiple requirements, for an example see Section 5.7.

A *design decision* records how the system will implement one or more requirements. The decisions recorded in a design decision can be very diverse, for example, decisions related to a technique, certain structure, or COTS product.

An *assumption* articulates an implicit (external) constraint. Non compliance to these constraints would yield an incorrect functioning system. The relation between the design decision and assumption should be documented very clearly. Assumptions can result in additional system requirements.

The concept of a *non conformance* explicitly documents exceptions in choices made earlier in the development process. All kind of reasons to do this can be thought of, e.g., when more information becomes available. Design decisions, assumptions, and non conformances are all part of the SDD's, and IDD's.

A *design rationale* records the argumentation for the design choice taken, but it can also be used to record the argumentation behind an assumption or non conformance. De-

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

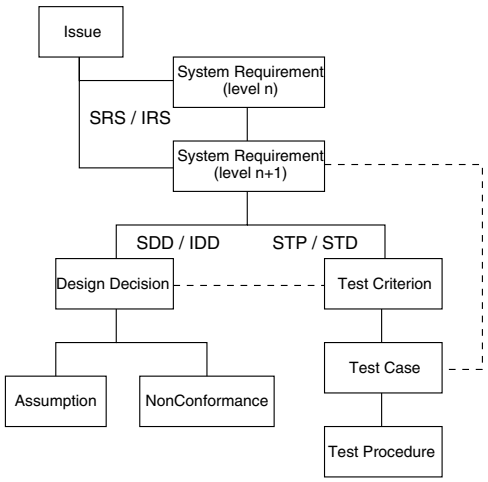

Figure 4. Requirement Traceability Model

sign rationales help to prevent endless discussions. An example of a design rationale is the argumentation for preferring a specific piece of middleware, in order to avoid high costs for licensing. Design rationales are defined as an attribute of a requirement and are *not* a requirement type.

The above types of requirements primarily relate to the design of the system. Similarly, LogicaCMG also defined requirement types for testing, namely *test criterion*, *test case*, and *test procedure*. A test criterion describes the conditions when a requirement has been successfully implemented in the system. It is possible to have more than one test criteria per requirement. A test case is an atomic test, which tests one or more test criteria. A test case describes the exact test situation, including preconditions, actions to be taken, and the condition under which the test case is successfully executed. Finally, the test procedure is a sequence of multiple test cases. The test cases and the test procedure are part of the STD's.

All aforementioned types of requirements are stored in the RMS as uniquely identifiable entities except for the design rationale, which is, as already mentioned, defined as an attribute of a requirement. Adding design rationales to the set of identifiable entities helps in two ways: it provides relations for traceability and it provides the necessary context for proper modifications.

The traceability relationships between the entities discussed above are summarised in Figure 4. The solid lines indicate the primary traceability links allowed to be set in the RMS and are n-to-n relations. Thus, each system requirement is linked to one or more design decisions, as well as to one or more test criteria. Likewise, each system requirement can be decorated with one or more issues concerning that requirement. The dashed lines give additional traceability links, that bypass the primary traceability relation. Thus, in principle test criteria come from system requirements, but in some cases a test criterion can be derived from a design decision as well. Likewise, a system requirement should in principle be accompanied by a test criterion, but in some cases it is easier to directly provide the test case instead. Although advisable, it is not always possible to come up with a test criteria for every system requirement.

5.3. Instantiating the RMS

In order to adopt RequisitePro (see Section 3.2) in the TMS setting, it needs to be configured so that the described document structure and the traceability model fit in. Moreover, its configuration should support the appropriate entities and relationships as occurring in the traceability model.

For the TMS case, the documents were divided into three levels. The first level includes the OCD, the SSS, and the SSDD documents. The second level includes all the SRS, and IRS documents. The final level includes the SDD, the IDD, and the STD documents. For every requirement attributes have been defined. Besides the obligatory internal id, also attributes such as design rationale, priority, status, and stability are defined in the tool. A particularly important attribute was the identifier provided for each requirement by the client – these identifiers provide the traceability to the requirements documents from the client. Thus, the internal identifier generated by the RMS was effectively ignored, and replaced by the client-provided identifier.

Observe that similar steps are required if another requirements management tools had been chose, e.g, DOORS from Telelogic.

5.4. Populating the RMS

After the RMS has been properly configured, it needs to be populated with the actual requirements. RequisitePro supports an interactive process in which paragraphs in MS Word documents can be marked as requirements, after which the type of requirement and the values for the corresponding attributes can be provided. In addition to that, traceability links to other requirements can be declared. Since the original MS Word documents have no strict structure, this process is largely manual, and hard to automate.

5.5. Updating the Requirements

Users of RequisitePro can modify requirements by navigating through the set of requirements and selecting a particular requirement for modification, which opens the requirement in the originating Microsoft Word document. Executing such an update, e.g., repair a typo or change a requirement statement, marks that particular requirement as changed. The traceability model now helps to trace the impact of that change through the rest of the system.

Unfortunately, in the TMS setting the requirements documents are owned by the client, and changes made by the vendor will lead to inconsistencies with the version maintained

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

by the client. Hence, changes to the requirements must come from the client, who provides a full new version of the requirements documents. The new set then must be analysed by hand, and changes to the underlying requirements and traceability links as stored in the RMS must be carefully executed. Although some help can be provided by taking the differences between two documents, this remains a cumbersome and error prone process.

5.6. Report Generation

LogicaCMG defined various reports to be generated from the RMS to support the various project team members. These reports are generated to support reviews, testing, design, and project management. Reports are defined for actors such as the project manager, test manager, designers, and requirements manager. Examples of these reports are generating an overview of all issues including status for the project manager, generating an overview of all system requirements covered by test procedures for the test manager, generating traceability matrices for the designer, and generating an overview of all requirements concerning a subsystem for the requirement manager.

The generated reports can be used for a basic form of systematic analysis or, for example, to verify the consistency, or correctness of requirements. RequisitePro offers limited functionality for this; if the offered support is insufficient separate software operating directly on the underlying database and the traceability links contained therein should be written (which amounts to bypassing RequisitePro).

5.7. Case Statistics

The outsourcer provided the requirements documentation which is decomposed into 1 SSS, 3 SRS, 9 IRS, and 3 IDD documents. Subsequently the outsource vendor used the traceability model of Figure 4 to structure the requirements from the SRS and IRS documents.

The development process involved only one iteration so far. During the first phase 160 issues emerged which had to be resolved with the client. Issues ranged from clarifications through inconsistencies and from adding and removing modules to restructuring of the modularisation. The development process particularly did not involve analysis of the received requirements, which potentially could have resolved a number of issues in an early stage of the development. An example of an issue is the following: One of the requirements explicitly mentioned the version number of a middleware component, however during design a different (older !) version of this component was believed to have significant advantages. The corresponding design decision thus had a different number than the originating requirement. The issue is easily resolved but has to be clearly communicated in order to keep the involved SSS consistent.

After the first phase the issues were fed back to the client. The subsequent contract renegotiations yielded a new set of SRS documents that roughly increased the number identified requirements by 80 and seriously modified 560 requirements. Figure 5 shows the statistics of the case study for the first phase, and Figure 6 shows the situation after the first iteration. As an example 120 system requirements that originate from an SRS were updated and 32 new ones were added. As a result the requirement database holds about 2400 related requirements. The system holds about 4700 traceability links.

Importing and updating the requirements from the respective SRS and IRS documents is a partly manual process as described in the previous subsection. The initial importation as well as the update after renegotiations both took 5–6 men days.

6. Discussion

Having described the way in which requirements management was conducted for the TMS system, we can now discuss selected observations, distill a number of key lessons learned. We propose a requirements engineering framework that addresses most of our identified concerns.

6.1. Observations

Importing and updating of requirements is related to change management. The import of the semi-structured requirements into the RMS is a troublesome process, whereas an automatic process is preferred. It takes quite some manual work to satisfy the requested quality. Although the input requirements are structured, they are not formalised. Diagrams, for instance, play an important role in clarifying requirements. But since they lack formalisation, diagrams have been left out of the requirements management system. Updating the set of requirements is an entirely manual process. It starts from the differential of the updated and initial version, changes are then updated in the RMS manually. The update process typically interferes with the tracing capabilities of the toolset; all links are invalidated, which makes the update a cumbersome and error prone process. This problem partly stems from the fact that the change management system does not distinguish between types of changes. Thus fixing a typo propagates through the traceability links in exactly the same manner as a drastic change such as decimating the available latency for some action. Categorisation or modularisation of types of changes should be supported by the RMS.

With respect to issue tracking and quality assurance, we observed that LogicaCMG uses issues to communicate with its outsourcer. Resolved issues are signed by both parties to protect them from potential legal conflicts. This process is satisfactory for both parties, although the problematic update

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

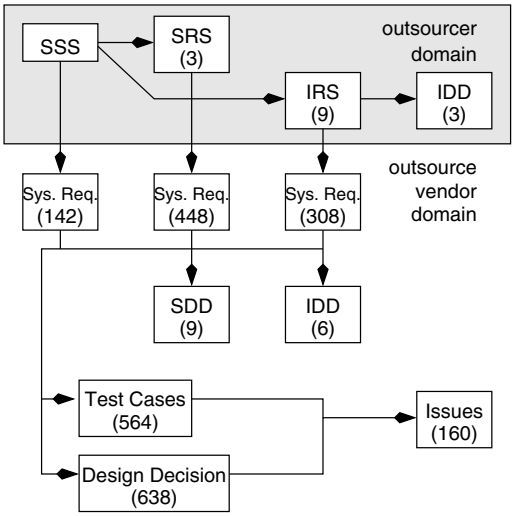

Figure 5. Phase I (s) indicates the number of requirements.

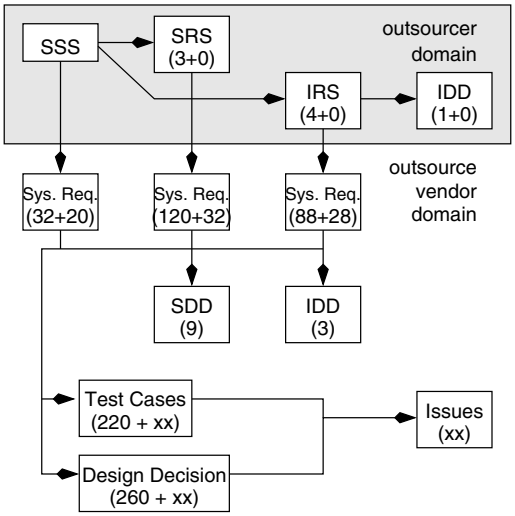

Figure 6. Phase II, (s + t) indicates s updated and t new requirements; xx is unknown.

inside the RMS remains. A risk exists that resolved issues are not correctly updated in the RMS, which renders the set of requirements inconsistent or even incorrect.

Status reports are generated from the RMS to support the communication with stakeholders. The reports are used for reviewing and analysing requirements for correctness, possible ambiguities, and consistency. The reports are used in project management for status tracking and issue tracking. The generation of these reports requires tailoring. The available tool, Rational SoDA, has too many limitations, such as poor quality reports, limited flexibility, and long processing times. To overcome these drawbacks, LogicaCMG develops a new tool which uses a more advanced query system than currently offered by RequisitePro.

As a consequence of this lack of proper reporting capabilities, developers use the navigation facilities of RequisitePro to locate information about requirements. The browser offers the opportunity to search through the complete set of requirements. Access control was not necessary to be implemented in this particular case study. All members of the development team within LogicaCMG are allowed the read and edit the complete set of requirements in the outsource vendor domain. The browser, although flexible, is not able to provide all required views. For example a view of the decomposition of a high-level requirement throughout the system shows the complete traceability path, whereas developers may only want to see part of it. Flexibility in generating views is essential, as views play a key role in the communication and interaction with stakeholders. This especially holds in an outsourcing context where clients have very specific demands for communication.

Tracking and tracing are important assets of an RMS. The traceability model helps to avoid a 'spaghetti' of traceability links. There is however the risk of inconsistency in the links, in part because of the manual process of providing the links. Links should also have a rationale that explains why a particular link exists. Making a link a first class element of the traceability model would solve this issue and potentially decreases the risk of an inconsistent set of requirements.

The current traceability model (Figure 4) supports detailed attributes for tracking. In practice, however, the status attribute of every requirement is not used. It is for LogicaCMG sufficient to report the status of a complete requirements document, e.g., a SRS.

The RMS has very limited support for analysis of requirements. Although the traceability model facilitates impact analysis, it lacks support for other types of analysis such as conformance checking. Conformance, and correctness checking are now implement by a review process. This again emphasises the importance of high quality report generation.

Flexibility of modularisation is an important feature of an RMS in the context of outsourcing. In our case study the requirements were already grouped into subsystems by the client. The outsource vendor executed the design step from SRS to SDD. During this design step developers objected to the initial modularisation. But, since the client owns the high-level design documents, their objections had to be negotiated with the client, which takes time. Three possible solutions are readily thought of: (1) the outsourcer hands over ownership of the high level design documents; (2) the high level design documents are discarded and issues are resolved in the domain of the RMS; or (3) the RMS supports flexible modularisations.

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

6.2. Lessons Learned

From our observations the following lessons should be taken in consideration when developing software systems in an outsourcing context:

1. Flexible modularisation and generation of views needs to be supported. Current tool support is not sufficient to satisfy these needs.

2. Explicit issue notes are an effective means for communication. Their role needs to be explicit in the requirements management process as well the tracking of these issues.

3. Tracking can be implemented effectively through an appropriate traceability model. Links between requirements in such a model must be first class elements.

4. Due to the distributed requirement engineering process in an outsourcing context, synchronisation of activities must be addressed explicitly. These activities include modifying and updating of the set of requirements.

6.3. A Conceptual Framework for Reconciling Requirements Evolution and Outsourcing

We conclude our discussion by proposing a conceptual framework of a Requirements Engineering System (RES) that implements the lessons learned. The primary purpose of this framework is to bridge the gap between the need for evolution of requirements, and the evolution impediments that are generated by the adoption of outsourcing. Parts of our proposed framework recur in existing commercial toolsets but none of them satisfies all features [7].

We refer to the proposed framework as a RES because typical specification activities such as analysis are also incorporated. The heart of our framework, depicted in Figure 7, is the requirements model with corresponding traceability model. In general this model is modularised according to a template, e.g. MIL-std 498 or IEEE-std 830-1998, used as document structure by the client. For every requirement entity attributes are defined and these entities are related according to the traceability model.

The requirements model allows for a structured analysis and updates through automatic tools. Dynamic modularisation is an important aspect of the proposed framework. The document structure is one modularisation, the (software) system decomposition is another. The underlying traceability model can generate multiple *views*, each taking a different perspective with corresponding partitioning and clustering of requirements and traceability links. This makes a view an ideal means for communicating issues with the outsourcer.

Traceability links themselves are first class entities in this model. Thus, in the traceability model the traceability links

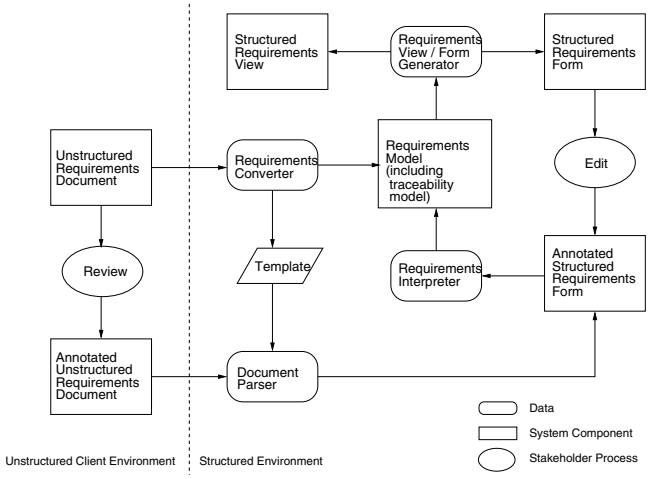

Figure 7. Proposed Requirements Engineering Framework

have a unique identifier as well as other attributes such as design rationale, and maybe even a description.

Another important aspect of the framework is the ability to transform the contents of the requirements model to an external structured document based on a template (e.g., by means of *forms*). Outsourcers mostly provide unstructured or semi-structured documents to the outsource vendor.

Incorporating the various changes originating from the unstructured environment poses a challenge for the structured environment. The outsource vendor and the customer operate in parallel, and in different development environments. Outsource vendors are not in the position to force outsourcers to adapt to their requirements engineering process, giving rise to the need for a conversion process. In our framework the conversion is done only once. After this conversion process requirements management and engineering is done in the structured environment. Therefore we need to reproduce a look-a-like of the initial unstructured document that is structured and can be edited. We refer to these as *forms*. To produce these forms a template is extracted during the conversion process, which is used to parse the annotated requirements documents, so the changes can automatically be imported into the RES. Forms are used to interact with the stakeholder, e.g., facilitate editing. Once edited, these changes can easily be interpreted and imported in the RES ensuring consistency of the set of requirements.

Views cannot be edited by the stakeholder and are primarily a means for communication and analysis. The process described is repeatable during the development life-cycle as well as during maintenance of the system and supports the management of evolving requirements.

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

6.4. Future Work

The proposed framework and the observations give way for several topics to investigate further in future research:

- How to determine the template (structure) of a semi-structured requirements document,
- How to specify the traceability model such that it has clear semantics,
- How to generate views for analysis, and conformance checking,
- How to interpret an annotated form,
- How can we parse an annotated document to a form according to the derived template, and
- Does this framework reduce the risks of introducing errors during evolution of requirements.

These research opportunities will be investigated in close cooperation with industry. LogicaCMG and other members of the MOOSE consortium provide a fertile experimental ground to arrive at answers to these questions.

7. Conclusion

In this paper we discussed how LogicaCMG has implemented requirements management in order to support requirements evolution for the traffic monitoring system, which LogicaCMG is implementing for an external customer. The customer took care of requirements elicitation and analysis, and decided to outsource the development to LogicaCMG. The focus of this paper is on analysing the implications of outsourcing on requirements management methods and tools.

We consider the following to be our key contributions. First of all, we discussed how requirement management was tackled in a fixed price contract between a client who created the requirements in the first place and an outsourcing vendor who was responsible for building the system. Relevant described results include the use of *issues*, the adopted traceability model, and the discussion of features that a requirements management system applied in an outsourcing context should have. We believe that sharing these experiences from LogicaCMG is valuable for both researchers and practitioners in the area of requirements management.

Second, we identified a number of problems with the requirements management methods and tools adopted. These problems concern the transition of requirements from the unstructured to the structured domain, the need to redo this transition upon requirements evolution, inadequate reporting facilities, and lack of sufficiently flexible modularisation support.

Last but not least, we proposed a framework that can be used as a research vehicle for addressing the concerns raised in this paper. Our framework facilitates multiple views, imports by means of forms and templates, and explicitly separates the structured from the unstructured environment. The elaboration of this model is the focus of our current research.

Acknowledgements We would like to thank the ITEA organisation for enabling and supporting the MOOSE collaboration. In particular, we would like to thank Robin Rijkers, and all the other team members of the TMS project at LogicaCMG for their cooperation and making this research possible.

References

[1] MOOSE. software engineering MethOdOlogieS for Embedded systems. `www.mooseproject.org`, 2004.

[2] Rasha Abbas, Philip Dart, Ed Kazmierczak, and Fergus O'Brien. Outsourcing software applications development: issues, implications, and impact. Technical Report 97/27, University of Melbourne, December 1997. 20 pages.

[3] Amer Al-Rawas and Steve Easterbrook. Communication problems in requirements engineering: a field study. In *Proceedings of the First Westminster Conference on Professional Awareness in Softare Engineering*, London, February 1996.

[4] Daniela Damian, James Chisan, Polly Allen, and Brian Corrie. Awareness meets requirements management: awareness needs in global software development. In *Proceedings of the ICSE International Workshop on Global Software Development (GSD 2003)*. University of Victoria, 2003.

[5] USA Department of Defence. Military standard on software development and documentation (mil-std-498), 1994.

[6] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First IEEE International Conference on Requirements Engineering*, pages 94–101, Colorado springs, April 1994.

[7] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: state of the practice. *IEEE Software*, 20(6):61–69, November–December 2003.

[8] Gerald Kontonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques*. John Wiley and Sons, 1998.

[9] Dean Leffingwell and Don Widrig. *Managing Software Requirements: A Unified Approach*. Addison Wesley, 2000.

[10] M. M. Lehman. Software's future: Managing evolution. *IEEE Software*, 15(1):40–44, January–February 1998.

[11] Päivi Parviainen, Maarit Tihinen, Marco Lormans, and Rini van Solingen. Requirements engineering: Dealing with the complexity of sociotechnical systems development. In José Luis Maté and Andrés Silva, editors, *Requirements Engineering for Sociotechnical Systems*. IdeaGroup Inc., 2004. Chapter 1. To be published.

[12] Rafael Prikladnicki, Jorge Audy, and Roberto Evaristo. Requirements management in global software development: Preliminary findings from a case study in a sw-cmm context. In *Proceedings of the ICSE International Workshop on Global Software Development (GSD 2003)*. University of Victoria, 2003.

[13] Karl E. Wiegers. Automating requirements management. *Software Development*, 7(7), July 1999.

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE