Proceedings

1st International Global Requirements Engineering Workshop GREW'07

GREW'07 – www.bth.se/grew07
Munich, Germany, August 27th, 2007

held in conjunction with the International Conference on Global Software Engineering ICGSE 2007 – www.icgse.org

Organizing Committee

Program Committee

Important Dates

Camera-ready papers due: July 27, 2007
Early Registration deadline: June 9, 2007 (www.icgse.org)
Workshop: August 27, 2007

Donors

Overview

Distributed multi-site software product development is increasingly becoming commonplace as companies become global not only in terms of customer base, but also with regards to large parts of the software product development that is spread over continents and cultures. Distribution is driven by that it enables companies to leverage their resources, and to draw on the advantage of proximity to customers and markets during large-scale software development.
The potential opportunities, however, come with new challenges for the product planning, management and development organizations of a company that affect requirements engineering of software products. The threat of defect increase and overruns in multi-site development has been documented in literature. According to industry experience reports, some of the main problems are attributed to heterogeneous understanding of requirements and substantial differences in domain understanding and interpretation. This is compounded by the fact that multi-site development usually is detrimental to informal communication between stakeholders such as product managers, experts, and developers, as these roles are often separated geographically.

The goals of this workshop is to identify, report, discuss, and address the challenges associated with requirements engineering (RE) and product management (PM) from two main perspectives:

(i) PM/RE for Global Software Development – assuring that the handling of requirements and products are effective and efficient in relation to a global/distributed development environment where development is conducted over multiple sites, and
(ii) Distributed PM/RE – the activities associated with PM and/or RE are conducted globally over multiple sites, and the development may be conducted in one single site or distributed around any number of sites.

Program Overview

The workshop will feature three sessions with seven papers presented. The workshop papers include submissions from both, industry and academia. There will be ample space for discussions on the session topics and the papers presented.

The session **Product Lines for Global Markets** addresses the definition, use and evolution of product lines in a multi-site distributed environment.

The session **Globally Distributed Communication** addresses requirements engineering and release planning in a distributed organization and the effects of factors like distance, communication mode and organizational structure on activities like negotiations and maintaining awareness of requirements.

The session **Challenges of Global Requirements Engineering: Consequences for Research** discusses challenges of distributed requirements engineering that have been elicited from IT professionals to build a basis for shaping future research in the global requirements engineering area.

Contact

For any further requests, please contact: tony.gorschek@bth.se

1st International Global Requirements Engineering Workshop GREW'07

WORKSHOP PROGRAM

(news and updates) www.bth.se/grew07 | www.icgse.org

Time and Place: Monday August 27th on location at the Technical University of Munich (TUM) (for room see update on www.bth.se/grew07)

Session	Papers	Time
Workshop opening	**Welcome and Introduction** (Tony Gorschek, Blekinge Institute of Technology)	09.30-09.40
Session: Product lines for global markets	**P1: Requirement Management in Software Product Lines** (Hyun Cho, Samsung Electronics Co. Ltd.)	09.40-10.00
	P2: Issue-based Variability Modeling (Anil Kumar Thurimella, Siemens AG, and Timo Wolf, Technical University of Munich)	10.00-10.20
(Session Chair) ***Gerald Heller*** *Hewlett-Packard GmbH*	*Session Topic Challenges (5-10min) and Open Discussion*	10.20-11.00
COFFEE BREAK		11.00-11.30
Session: Globally distributed communication (part I)	**P4: The Effects of Communication Mode on Distributed Requirements Negotiations** (Teresa Mallardo, Fabio Calefato and Filippo Lanubile, University of Bari, and Daniela Damian, University of Victoria)	11.30-11.50
	P5: The Effects of Distance, Experience, and Communication Structure on Requirements Awareness in Two Distributed Industrial Software Projects (Irwin Kwan, Daniela Damian and Sabrina Marczak, University of Victoria)	11.50-12.10
(Session Chair) ***Daniela Damian*** *University of Victoria*	*Session Topic Challenges (5-10min) and Open Discussion*	12.10-13.00
LUNCH		13.00-14.00
Session: Globally distributed communication (part II)	**P3: A Model of Requirements Engineering at Organizational Interfaces: An Empirical Study on Distributed Requirements Engineering** (Dorina-C. Gumm, University of Hamburg)	14.00-14.20
	P6: Release Planning in Distributed Projects (Korbinian Herrmann, Technische Universität München)	14.20-14.40
(Session Chair) ***Samuel Fricker*** *ABB and University of Zurich*	*Session Topic Challenges (5-10min) and Open Discussion*	14.40-16.00
AFTERNOON TEA		16.00-16.20
Session: Challenges of global requirements engineering: consequences for research	**P7: The Challenges of Distributed Software Engineering and Requirements Engineering: Results of an Online Survey** (Timea Illes-Seifert and Andrea Herrmann, Universität Heidelberg, and Michael Geisser and Tobias Hildenbrand, Universität Mannheim)	16.20-16.40
(Session Chair) ***Andrea Herrmann*** *University of Heidelberg*	*Session Topic Challenges (5-10min) and Open Discussion*	16.40-17.20
Workshop summary	**Summary of Challenges and Issues Identified: Workshop Feedback and Closing** (Tony Gorschek, Blekinge Institute of Technology)	17.20-17.50

Sessions will consist of paper presentations (15 min), and a short Q/A (~5 min) for each paper. After paper presentations the Session Chair holds a short (~5 min) overview of the session topic challenges as an introduction to the session discussion (~60 min/per session).

1st International Global Requirements Engineering Workshop (GREW´07), Munich, Germany, August 27th, 2007.
(Held in conjunction with the International Conference on Global Software Engineering ICGSE)

Table of Contents

1st International Global Requirements Engineering Workshop (GREW'07)

1st International Global Requirements Engineering Workshop (GREW´07)

Tony Gorschek
Blekinge Institute of Technology
School of Engineering,
Department of Systems and Software Engineering
tony.gorschek@bth.se

Samuel Fricker
ABB Switzerland Ltd.
samuel.fricker@ch.abb.com
University of Zurich, Department of Informatics
fricker@ifi.unizh.ch

Abstract

GREW´07 brings researchers and industry practitioners together to discuss the area of global product development from a requirements engineering and product management perspective. The workshop aims at analyzing selected challenges, which are put forward by accepted papers, in detail., The session discussions then lift the view in an attempt to identify the future needs with regards to research and study.

Industry presence at the workshop is intended to ground the discussions of future research, helping to assure relevance and usefulness from both an industrial and an academic perspective.

1. Introduction

Distributed multi-site software product development is increasingly becoming commonplace as companies become global not only in terms of customer base, but also with regards to large parts of the software product development that is spread over continents and cultures. Distribution is driven by that it enables companies to leverage their resources, and to draw on the advantage of proximity to customers and markets during large-scale software development [1].
The potential opportunities, however, come with new challenges for the product planning, management and development organizations of a company that affect requirements engineering of software products. The threat of defect increase and overruns in multi-site development has been documented in literature. According to industry experience reports, some of the main problems are attributed to heterogeneous understanding of requirements and substantial differences in domain understanding and interpretation. This is compounded by the fact that multi-site development usually is detrimental to informal communication between stakeholders such as product managers, experts, and developers, as these roles are often separated geographically.

2. Workshop Goals

The goals of this workshop is to identify, report, discuss, and address the challenges associated with requirements engineering (RE) and product management (PM) from two main perspectives:
(i) PM/RE for Global Software Development – assuring that the handling of requirements and products are effective and efficient in relation to a global/distributed development environment where development is conducted over multiple sites, and
(ii) Distributed PM/RE – the activities associated with PM and/or RE are conducted globally over multiple sites, and the development may be conducted in one single site or distributed around any number of sites.

2.1. Target Groups

The workshop is relevant for any researcher or professional that is faced with the challenges of conducting requirements engineering or product management in a global environment or of working in a distributed organization. The workshop focuses on, but is not limited to, a pre-solution implementation perspective, as the initial requirements selection, analysis, refinement, and communication activities are of particular importance in a global environment. This perspective, unlike the project centered, is rarely covered in software engineering. In addition, facets of economy and management of the software engineering endeavor are also highly relevant.

3. Workshop Sessions

The GREW’07 workshop features following sessions (see www.bth.se/grew07 for details).

3.1. Product Lines for Global Markets

This session addresses the definition, use and evolution of product lines in a multi-site distributed environment.

Session Chair: Gerald Heller, Hewlett-Packard GmbH.

Paper 1: Requirement Management in Software Product Lines (Hyun Cho, Samsung Electronics Co. Ltd.).
ABSTRACT: Product line requirement (PLR) secures the success of product line adoption by giving direction to the development and management of product line architecture and core asset. But, it is not easy to develop PLR to leverage requirements reuse and to serve design inputs while expressing variations of every product requirements in simple and flexible way. This paper introduces how PLR are categorized and modeled according to the characteristics of requirements and how they are structured with their relevant artifacts.

Paper 2: Issue-based Variability Modeling (Anil Kumar Thurimella, Siemens AG, and Timo Wolf, Technical University of Munich).
ABSTRACT: Product line development is tending to be globally distributed with complicated organizational models, involving distributed expertise from diverse cultures. Therefore the use of the communication models for variability management is a vital issue. SYSIPHUS supports informal collaboration in software engineering by using the issue model (a rationale model) as a communication model. This paper proposes that the issue model of the SYSIPHUS can be used to model product line variability supporting informal collaboration for the variability management, which paves the way for a new concept called issue-based variability modeling. Further issue-based variability modeling addresses the capture of rationale, and supports the instantiation and evolution of the variation points. This paper is illustrated using an industrial case study from the domain of infotainment systems and is evaluated empirically.

3.2. Globally Distributed Communication

This session addresses requirements engineering and release planning in a distributed organization and the effects of factors like distance, communication mode and organizational structure on activities like negotiations and maintaining awareness of requirements.

Session Chair (part I): Daniela Damian University of Victoria.

Paper 1: The Effects of Communication Mode on Distributed Requirements Negotiations (Teresa Mallardo, Fabio Calefato and Filippo Lanubile, University of Bari, and Daniela Damian, University of Victoria)
ABSTRACT: Videoconferencing is generally considered as the most appropriate medium to conduct requirements negotiations between remote stakeholders. To improve the effectiveness of distributed requirements negotiations, drawing upon the postulates of theories on media selection, we argue that a combination of lean and rich media is needed. In this paper we empirically test the hypothesis that the early resolution of uncertainties through an asynchronous lean medium can shorten the list of open issues to be negotiated over a synchronous rich channel.

Paper 2: The Effects of Distance, Experience, and Communication Structure on Requirements Awareness in Two Distributed Industrial Software Projects (Irwin Kwan, Daniela Damian and Sabrina Marczak, University of Victoria).
ABSTRACT: In global software development, communication is difficult due to distance between sites. How effectively do team members distributed among multiple geographical locations become aware of changes and clarifications to requirements? In a case study of two different global software development projects, we observed how requirement analysts, developers, and testers maintain awareness of changes in the project. To gather data, we attended local and remote meetings, and conducted interviews of project team members. Based on our experience with these projects, we discuss the following awareness factors in software development: distance, experience of team members, and communication structure. We present the effects on awareness, and provide some lessons learned for global software development projects. We expect these lessons learned can be used by projects with similar settings.

Session Chair (part II): Samuel Fricker ABB Switzerland Ltd. and University of Zurich.

Paper 3: Requirements Engineering between Organizational Units (Dorina-C. Gumm, University of Hamburg).
ABSTRACT: In this paper results are presented from an empirical study about the relationship between requirements engineering practice and distributed software project settings. The focus here lies on organizational distribution and the requirements engineering activities that take place between organizational units. In order to understand

distributed requirements engineering, the concept of organizational interfaces is introduced. Requirements engineering activities are then analyzed with respect to organizational interfaces. The resulting model aims at facilitating practitioners and researchers to design distributed processes and understanding respective challenges.

Paper 4: Release Planning in Distributed Projects (Korbinian Herrmann, Technische Universität München).
ABSTRACT: During release planning project managers decide what to deliver in a specific release. In globally distributed projects his decision is a "wicked problem" [4] because local decisions might contradict global decisions. Project managers have to respect many factors such as customer preferences, time and budget as well as constraints from development. Existing approaches to release planning neglect the influence of system models. This paper proposes a single tool that supports both release planning and modeling in globally distributed software projects. The approach allows developing a single model for every release. This enables project managers to make an informed decision with respect to the system model during release planning.

3.3. Challenges of Global Requirements Engineering: Consequences for Research

This session discusses challenges of distributed requirements engineering that have been elicited from IT professionals to build a basis for shaping future research in the global requirements engineering area.

Session Chair: Andrea Herrmann University of Heidelberg.

Paper 1: The Challenges of Distributed Software Engineering and Requirements Engineering: Results of an Online Survey (Timea Illes-Seifert and Andrea Herrmann, Universität Heidelberg, and Michael Geisser and Tobias Hildenbrand, Universität Mannheim).
ABSTRACT: Growing globalization and increasing complexity of software lead to international and national collaboration of geographically distributed organizations, sites and persons. Therefore, it becomes more important to understand and to know how to optimize distributed software development. Thus, we performed a survey among professionals on their experiences with distributed software development. This publication presents an evaluation of 744 questionnaires, with a focus on requirements engineering. The survey results show that a variety of human and process-related aspects are important for distributed software development. They furthermore emphasize the importance of communication in equirements engineering: Communication, particularly face-to-face meetings, represents the most frequently mentioned solution to diverse problems. Similar results were found before, but this survey supports them with a high quantity of data.

4. Workshop Organization

4.1. Organizing Committee

- Tony Gorschek, organizing chair (Blekinge Institute of Technology, Sweden),
- Aybüke Aurum (University of New South Wales, Australia),
- Christof Ebert (Vector Consulting, Germany),
- Samuel Fricker (ABB Switzerland Ltd., Switzerland, and University of Zurich, Switzerland),
- Conny Johansson (Ericsson, Sweden and Blekinge Institute of Technology),
- Claes Wohlin (Blekinge Institute of Technology, Sweden).

4.2. Program Committee

- Ayse Bener (Boğaziçi University, Turkey),
- Maya Daneva (University of Twente, the Netherlands),
- Alan M. Davis (University of Colorado at Colorado Springs, USA),
- Robert Feldt (Blekinge Institute of Technology, Sweden),
- Samuel Fricker (ABB Switzerland Ltd., Switzerland, and University of Zurich, Switzerland),
- Paul Grünbacher (Johannes Kepler Universität (JKU) Linz, Austria),
- Gerald Heller (Hewlett-Packard, Germany),
- Andrea Herrmann (University of Heidelberg, Germany),
- Ann M. Hickey (University of Colorado at Colorado Springs, USA),
- Natalia Juristo (Universidad Politécnica de Madrid, Spain),
- Irwin Kwan (University of Victoria, Canada),
- Filippo Lanubile (University of Bari, Italy),
- Marek Leszak (Alcatel-Lucent, Germany),
- Sabrina Marczak (University of Victoria, Canada),
- Michael Mattsson (Blekinge Institute of Technology, Sweden),
- Audris Mockus (Avaya Labs Research, USA),
- Michael Ochs (Fraunhofer IESE, Germany),
- Peter Sawyer (Lancaster University, UK),
- Darja Smite (University of Latvia, Latvia),
- Giancarlo Succi (Free University of Bolzano-Bozen, Italy),
- Richard Torkar (Blekinge Institute of Technology, Sweden)

Requirement Management in Software Product Line

Hyun Cho
Samsung Electronics Co. Ltd.
hcho@samsung.com

Abstract

Product line requirement (PLR) secures the success of product line adoption by giving direction to the development and management of product line architecture and core assets. But, it is not easy to develop PLR to leverage requirements reuse and to serve design inputs while expressing variations of every product requirements in simple and flexible way. This paper introduces the types of requirement variations, the scheme of PLR specification and the relationships of PLR and other requirements.

1. Introduction

PLR is a medium for capturing and communicating to all interested parties what is needed in the entire product line. PLR provides a basis for most of the management and engineering functions associated with maintaining the product line, such as assessing proposed changes, resolving requirements disputes, developing test requirements, writing manuals, planning support activities and implementing enhancements. In addition to its role as a traditional requirements vehicle, it must capture the commonality and variability between the individual products that comprise the product line.

Since Parnas introduced commonality and variability for interface design [1], many researchers have been expanded this concept to software development process [2], requirement engineering [3][[4][5], design [6][7][8][9], and others. In addition, some researches tried to define the category of variability [10][11], the abstraction levels of variability [12], and the variability management[13].

Although a bunch of good methods and guidelines was introduced and practiced, capturing commonality and variability cannot be achieved unless the form and content of PLR enables it to be understood by all of its potential stakeholders: strategic planning, product planning, product development, engineering management, engineering development, testing organizations, documentation, etc. These participants have different perspectives and interests of varying scope and depth and PLR must be capable of addressing these different views.

This paper describes how to represent, organize, and document requirements effectively for entire product lines, suitable for use in developing using a Product Line development approach

2. PLR Specification

In requirements not designed for describing entire product lines, the organization of the requirements into sections and subsections provides a logical grouping that is intended to facilitate locating and understanding the requirements. In PLR, this is also the case, but there is an additional factor to consider - the variations between products in the product line.

It is important to remember that the requirements specification of product lines should describe the requirements of the core assets that are to be developed. If the core assets need to support different features or combinations of features, PLR needs to explicitly specify these variations and provide an indication of what combinations are valid. Such variations generally take four flavors: Mandatory, Optional, Alternative, and Variable. The following sections describe what these types of requirements are and how to represent them.

2.1. Variation Representation

The basic concepts or types of variation can be also applied to PLR specification. Table 1 provides an overview of requirement types and the criteria for inclusion of a requirement in a product line instance.

Table 1. Variation types of requirement

Types	Meaning
Mandatory	The requirement should be included to all product lines with no variants.
Alternative	The requirement can be mutual exclusively selectable among the feature group.
Optional	The requirement can be selected multiple in a feature group
Variable	The requirement can take option values as a variant

The three variation types, Mandatory, Alternative, and Optional, are widely used to model variants of product line. But, Variable is newly introduced variation type to manage variants that come from diversity of each option value such as throughput, response time, number of buttons, and etc. As Variable is not type of representation for the inclusion or exclusion of a grouping of requirements, it can be combined with other variation types, Mandatory, Optional, and Alternative.

For illustration purpose, the requirements of Weather Station products are taken to show how PLR specification looks like and how PLR is differ from product requirements (PR). Let say a Weather Station consists of some form of user interface and some number of devices that provide weather-related measurements, such as temperature and barometric pressure. Some products in the product line will have very simple user interfaces, for example a very small LCD display with associated control buttons, while others will be more complex, for example, with larger touch-screen displays.

When a statement is written for a product like below, it might be true as a requirement of a product

- *The weather station shall measure barometer.*

In PLR perspective, this statement is always true if all products include this feature. Then, this statement can be mandatory requirement. On the contrary, this statement may not be true if some products do not support this requirement. Hence, the statement should be changed to achieve specification correctness throughout product line. The new requirement specification need to be changed to choose whether to support time display or not and it might look like

- *The weather station shall be capable of being built with or without barometer.*

This is typical example of optional requirements.

Alternative is redefined as a special case of Optional requirements, that is, the selection of one alternative from many optional requirements. Thus, each of the individual options is flagged as optional, meaning that each product must choose whether to include the requirements grouping or not and then the additional constraint is applied to choose option mutually exclusive from these groupings. Hence, Alternative can be expressed like below

- *The system shall be capable of being built with or without thermometer.*
- *The system shall be capable of being built with or without barometer.*
- *The system shall be capable of being built with or without hygrometer*
- *The system shall be capable of being built with support for exactly one of thermometer, barometer, or hygrometer*

Finally, Variable, used to specify the variation of number, range or set across product line, can be specified like below.

The weather station shall be capable of being built with or without thermometer, which is able to measure temperature with a certain percentage of precision number

If PLR should have forms like previous examples, PLR specification would be very difficult to write, read, and understand. To facilitate understanding and the use of PLR for actual product requirements, it is desirable to write the actual requirements statements as if there were no variability and denote the variability via separate means.

For this purpose, some attributes are introduced and Table 2 shows how attributes are used to model PLR. In order to represent all the variation types, two attributes, pline_Applicability and pline_Grouping, are introduced. pline_Grouping effects only if pline_Applicability has Optional. Alternative is implicitly represented by having pline_Applicability attribute with a value of "Optional" and pline_Grouping attribute with a value of ChooseOne.

For Variable type, variable name is enclosed with % symbols in both side of variable and handled in the same manner as DOS shell variable substitutions. In addition, Variable type requires some attributes to explain the variable and to provide information about the values that it can take. Thus, three attributes, pline_VariationCardinality, pline_VariableRange, and pline_VariableType, are defined to represent cardinality of value, value range, and value type respectively.

Table 2. Attributes for product line requirement representation

Attribute	Type	Description
pline_Applicability	Enum { Comment | Mandatory | Optional | Variable }	Comment: The item is not requirement but comment Mandatory: The requirement group is mandatory within its context. Optional: The requirement group is optional within its context. Variable. The item represents a variable definition.
pline_Grouping	Enum { All | ChooseOne | ChooseZeroOrOne | ChooseAtLeastOne }	All. All requirements are included. ChooseOne. Only one requirement is included from the requirements group. ChooseZeroOrOne. At most one requirement is included from the requirements group. ChooseAtLeastOne. At least one requirement is included from the requirements group.
pline_VariationCardinality	String with syntax	Specify how many values of the variable type must or may be provided by the product definition. It must be single, non-negative integer or a range of the form n..m where n and m are non-negative integers ($n \leq m$)
pline_VariableRange	String with syntax	Applies only to requirements whose pline_Applicability is Variable. It forms like {n..m} where n and m are minimum and maximum value respectively.
pline_variableType	Enum{ Integer | Real | Enumeration | Set | Custom }	Applies only to requirements whose pline_Applicability is Variable.

Hence, PLR of Weather Station can be rewritten like blow.

```
1. Description [pline_Applicability=comment]
1.1. The products in the product line consist of Weather Stations. A Weather Station consists of some form of user interface and some number of devices.

2. Measurement [pline_Applicability =mandatory]
2.1.   Thermometer [pline_Applicability=mandatory]
2.1.1 The system shall be capable of measureing temperature
2.2   Addltioanl Measurement [pline_Applicability =optional, pline_Grouping=ChooseOne]
2.2.1. Barometer [pline_Applicability =optional]
2.2.1.1. The system shall be capable of measuring atmosphere pressure
2.2.2   hygrometer [pline_Applicability=optional]
2.3.2.1. The system shall be capable of measuring humidity.

3. Network Support [pline_Applicability =optional]
3.1. Network Card [pline_Grouping =mandatory]
3.1.1. The system shall support %NETWORK_SPEED%. NIC card.
3.1.1.1 NETWORK SPEED [pline_Applicability = Variable;
                       pline_VariableCardinality = 1..2;
                        pline_VariableType = Enum;
                       pline_VariableRange = 10/100T, 1G ]
```

2.2. Dependency Representation

When PLR groupings are nested, the meaning matches the intuitive interpretation that any sub-groupings inside an optional grouping are included only if the enclosing grouping is also included. In other words, if an optional grouping is not included, then all sub-groupings are also not included. This provides a natural way to express dependencies between optional features simply using the organization of the requirements.

However, it is not always possible to represent all relationships between requirements groupings in a hierarchical or nested manner. In some case, a requirement may have multiple dependencies with other requirements group. To represent this multiple dependency, the attribute, pline_Requires, is defined and the dependency is defined by listing dependent requirements as a comma separated values. For example, time synchronization depends on network support and time server, it can expressed like

```
2.2. Synchronization [pline_Applicability =optional]
                     [pline_Requires="Network Support,
                                      Time Server"   ]
```

if the requirement is further refined to having time server feature.

The rules for dependency control of optional requirements groups mentioned above are the most common to ruling dependencies between different groups. However, this is not all inclusive and does not even address all of the reasonable situations that might arise, let alone all possible situations.

To covering all possible situations, pline_IsPresent is defined and it borrows the concepts of Java-like expression where the terms are references to requirement groups. In pline_IsPresent, standard Java-like precedence and ordering applies, and Java operators and parentheses may be used as necessary. The result of the expression must be a Boolean value and the requirement is considered for inclusion if the pline_IsPresent expression evaluates true. Otherwise it is not included in the requirement group. In addition to referencing requirement groups, one additional keyword is allowed, SELECTED. This is replaced with the current selection state for the requirement group in question. The example of pline_IsPresent looks like

```
2.2. Synchronization [pline_Applicability =optional]
                     [pline_IsPresent=”Network Support &&
                                      Time Server &&
                                      SELECTED”]
```

Table 3 shows some examples of pline_IsPresent expressions and their meanings.

Table 3. Example of pline_IsPresent expression

Expression	Meaning
Network Support && Time Server && SELECTED	The requirement is available for selection if “Network Support” and “Time Server” are both presented.
Network Support && Time Server	The requirement is automatically included if “Network Support” and “Time Server” are both presented.
(Network Support || Time Server) && SELECTED	The requirement is available for selection if either “Network Support” or “Time Server” is presented.
Network Support || Time Server	The requirement is automatically included if either “Network Support” or “Time Server” is presented.

As the semantic of each pline_IsPresent is slightly different, the expression should be carefully described to work with original intent.

3. Requirements Organization and Management Process

3.1. Requirement Structure

As a result of PLR introduction, requirement documents are reorganized like Figure 1 to leverage requirements reuse and to manage requirements efficiently.

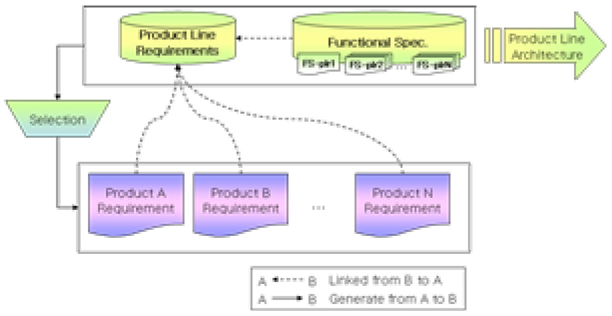


Figure 1. Requirement Structure

As described in previous sections, PLR is a collection of all PR including future product requirements and plays a centerpiece of all links between PLR, PR and FS. This link help to trace how the PLR is varied in the PR and how PLR is designed, implemented, and tested.

Functional Specifications (FS) are collections of behavior specifications of each requirement in PLR. FS precisely describes the behaviors and constraints of PLR and used inputs of product line architecture construction and core asset development.

Product Requirements (PR) is a requirement of specific product development and it is instantiated from PLR.

The selection is compulsory process for PR instantiation from PLR and the detail selection process is described in 3.2.

3.2. Requirement Instantiation

Program Manager (PM) creates a project file of PR (PPR) for a new product which specifies requirements of base model and its series. Different from PR, PPR does not contain actual product requirements. Instead, it lists just information of selected requirements among optional and alternative and specific values for variable requirements. PPS is created through product selection tool, which is front-end tool of PLR and guides user to create PR by showing available requirements, which are determined through the analysis of PLR attributes and dependencies of the selected requirements. With the use of PPR, every PR can be reproducible by mapping PPR information to PLR and PM does not need to distribute big and large PR documents.

Once PPR is created, PM distributes PPR to regional marketing representative (RMR), organized worldwide to promote new products and to collect regional customer’s requirements. Each RMR reviews

PPR and return it to PM. Then PM merges all PPR and requests Requirement Manager (RM) to open review meeting with the member of Change Control Board (CCB). The members are architect, development team, SQA team and they work at geographically distributed regions. Due to geographical distribution and time difference, teleconference equipments, which integrates video, phone, and web technology, is used to sharing opinions of review participants.

When CCB agrees to the contents of PPR, the first official release of PR is created as a form of document and maintained as a project artifact.

3.3. Requirement Change Management

The process and activities of change management in product line environment are similar to those of requirement instantiation except the processes is invoked by the submission of change request (CR) and RM facilitates all change process. When CR is submitted from stakeholders and CCB reviews and approves CR via integrated teleconference. Generally, CCB manages the changes of PR but, in product line environment, CCB has to manage one more type of change. That is PLR. The difference of these two changes is rooted from the change sources.

For PR, the sources of changes are same to those of traditional product development and CR of PR is handled similar ways of normal product requirement change process except some CRs could be escalated to CR of PLR. Usually, CR that affects architecture and/or requirement variations is usually escalated to CR of PLR.

On the other hand, CR of PLR largely comes from the changes of product and technology roadmap. Both roadmaps describe brief requirements of each feature in roadmap and outlook of release. In the product-wise development paradigm, the roadmap is maintained in the separated documents and does not belong to any development artifacts until they are included development project. But, in product line approach, the requirements of roadmap are specified in PLR whenever they are introduced and are valuable inputs to manage requirements variation and to plan product line architecture and core asset development.

4. Conclusion

As know as the most common reasons for project failure are rooted in poor estimation or weak definition of requirements. Likewise, PLR also important to planning, managing, and developing product line architecture and core assets which are critical factors to the success of product line approach.

In this paper, tow major aspects of PLR are mentioned; PLR specification and management.

For PLR specification, seven attributes are defined to handle all variation types and dependencies in PLR. PLR is quite a well structured with the attributes and become more extendible and flexible by changing attribute contents or adding new attributes.

For management perspective, some processes and activities are attached additionally to manage PLR. These changes are slight to the existing requirement management process but these are so important to keep the integrity of PLR throughout all kinds of change requests.

5. References

[1] D.L. Parnas, "On the Criteria to Be Used in Decomposing Systems into Modules," Comm. ACM, vol. 15, no. 12, pp. 1053-1058, Dec. 1972.

[2] David M. Weiss, Chi Tau Robert Lai. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, 1999.

[3] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson., "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[4] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe. Feature-Oriented Product Line Engineering. IEEE Software, 19(4):58–65, July/August 2002.

[5] M. Griss, J. Favaro, and M. d'Alessandro, Integrating Feature Modeling with the RSEB. In Proceedings of the Fifth International Conference on Software Reuse, pages 76–85, Vancouver, BC, Canada, June 1998.

[6] Hassan Gomaa and Michael Eonsuk Shin. Multiple-view meta-modeling of software product lines. In Eighth International Conference on Engineering of Complex Computer Systems, December 2002.

[7] Isabel John and Dirk Muthig. Product Line Modeling with Generic Use Cases. In Proceedings of the Second Software Product Line Conference, August 2002.

[8] Matthias Clauss. Generic modeling using uml extensions for variability, In OOPSLA 2001 Workshop on Domain Specific Visual Languages, September 2001.

[9] Gomaa, H. Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures, Addison-Wesley, July 2004.

[10] Felix Bachmann, and Len Bass. Managing Variability in Software Architecture, Proceedings of the ACM SIGSOFT

Symposium on Software Reusability (SSR'01), pages 126–132, May 2001.

[11] Günter Halmans, and Klaus Pohl, Communicating the variability of software-product family to customers, Proceedings of the Software and Systems Modeling, volume 2, pages 15–36. Springer, February, 27, 2003

[12] Mikael Svahnberg and Jan Bosch, Issues Concerning Variability in Software Product Lines, In Third International Workshop on Software Architectures for Product Families, pages 146–157, 2000.

[13] Ivar Jacobson, Martin Griss, and Patrik Jonsson., Software Reuse: Architecture, Process and Organization for Business Success. Addison-Wesley, 1997.

Issue-based Variability Modeling

Anil Kumar Thurimella
Siemens AG, CT SE1
Otto-Hahn-Ring 6
Munich, Germany
thurimel@in.tum.de

Timo Wolf
Chair for Applied Software Engineering
Technical University of Munich
Munich, Germany
wolft@in.tum.de

Abstract

Product line development is tending to be globally distributed with complicated organizational models, involving distributed expertise from diverse cultures. Therefore the use of the communication models for variability management is a vital issue. SYSIPHUS supports informal collaboration in software engineering by using the issue model (a rationale model) as a communication model. This paper proposes that the issue model of the SYSIPHUS can be used to model product line variability supporting informal collaboration for the variability management, which paves the way for a new concept called issue-based variability modeling. Further issue-based variability modeling addresses the capture of rationale, and supports the instantiation and evolution of the variation points. This paper is illustrated using an industrial case study from the domain of infotainment systems and is evaluated empirically.

1. Introduction

A software product line is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1]. Using product line approaches software products can be deployed faster, cheaper and quicker. The software product line engineering [2] paradigm includes two processes: domain engineering and application engineering as defined in [2]. Variability management is an important aspect of software product line engineering. In many industrial domains (such as automotive, mobile phone, domestic appliances etc), mass customization is increasing due to increasing number of customers and therefore the complexity of variability is growing. In addition, the increasing global distribution of mass customization, domain and application engineering, as well as the distribution of expertise motivates the research for global software product line engineering.

Requirements engineering is a critical phase of software engineering and is complicated, because of the communication problem between stakeholders. Product line requirements engineering has additional complexity of variability management and becomes even more complex. Communication models that provide strong communication and collaboration between stakeholders are vital in the context of global requirements engineering, and address the problems that results from outsourcing and globalization. In global product line requirements engineering communication models are needed for both: requirements and variability management. When the collaboration between the stakeholders takes place without concrete negotiation rules, it is termed as informal collaboration.

Rationale is the justification behind decisions, including alternatives explored, evaluation criteria, arguments and decisions [4]. The rhetorical Questions, Options and Criteria (QOC) [5] model is a way of capturing rationale in the form of a graph. Rationale improves the understandability of requirements. Therefore it is very useful for global requirements engineering, in which requirements must be understood in various cultures. Product lines are long-term investments and are to be maintained over a longer period of time (compared to products). As rationale supports the evolution of requirements [4], investments of applying rationale management can pay off.

Variability management is the essence of the current product line practices and is based on variability modeling. The current variability modeling techniques [2, 7 and 8] capture the variation points using dependencies and constraints. The major activities of the variability management are: variability identification, variability modeling, product instantiation (instantiating variations for specific products in application engineering) and variability evolution. All these activities involve the collaboration of stakeholders. The major variability management problems identified by this paper are:

Problem1. Under specified representation: The current variability modeling techniques such as OVM [2], feature modeling (e.g FODA [8]) and the use case extensions by Gomma [7] do not answer the following questions and thus, they are underspecified:

- Why is a dependency mandatory, optional or alternative?
- Why is a constraint requires or excludes?
- How is the variability related to mass-customization?

The understandability of these underspecified variability models gets worse in the global context, as the stakeholders have regional, cultural and linguistic differences. Further these variability models do not provide information for product instantiation and variability evolution. Therefore we claim that capturing rationale information for variability models is critical.

Problem 2. Gap between domain and application engineering: In large product line organizations, domain engineering is realized as a separate organizational unit with respect to application engineering units. Due to this organizational separation their roles, clients, timelines, milestones and goals are different thus crating a gap between domain and application engineering. Domain engineering teams can be concerned with identifying and creating variation points, while application engineering teams are instantiating the variation points. Due to this gap, it may be sometimes easier to develop a domain requirement from the scratch instead of instantiating the appropriate variation points. But reducing the requirement redundancies among products is important for the product line maintenance. Increasing numbers of application engineering teams makes this gap worse. To reduce this gap, we are researching on variability modeling techniques that support both domain engineering (e.g. identification and evolution) and application engineering (instantiation of variation points).

Problem 3. Lack of collaboration for variability management: The variation points in application requirements engineering are to be instantiated by the collaboration of domain engineers, application engineers and customers, and therefore the product instantiation is a collaboration intensive problem. The existing product instantiation approaches such as [9], [10], [11] and [12] and do not address the collaboration between stakeholders. Further the collaboration in the variability identification and evolution are not addressed as well.

Section 2 of this presents the background information required for this paper. Section 3 shows related work. The infotainment systems are introduced in section 4. Section 5 proposes the issue-based variability modeling which is the core contribution of this paper. In section 6, variability management activities such as variability identification and capture of rationale, product instantiation and variability evolution are addressed. Section 7 gives empirical evaluations and the conclusion of this paper is given in section 8.

2. Background information

This section provides the concepts that are vital ingredients for building up the meta-model of issue-based variability modeling. Section 2.1 presents the meta-model for the Orthogonal Variability Model (OVM), section 2.2 presents SYSIPHUS and 2.3 presents Rationale-based Variability Management in Requirements Engineering (RVMRE).

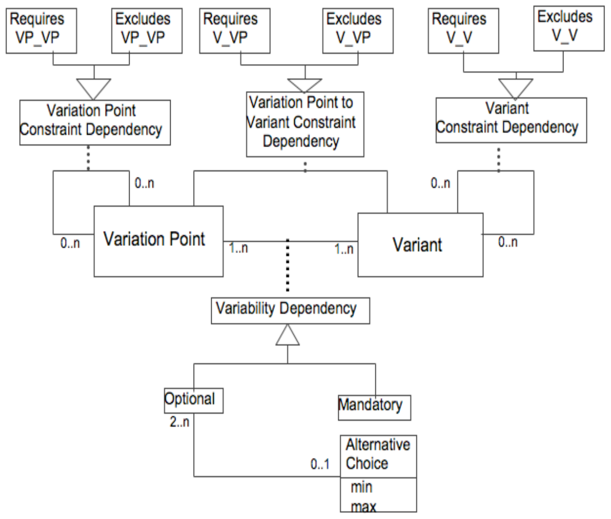


Figure 1: Variability meta-model

2.1 Variability meta-model

The Orthogonal Variability Model (OVM) [2] represents variability and advocates the use of a centralized and separate variability model for all models of the software development process, such as use case diagrams, class diagrams, activity diagrams, component diagrams etc. In OVM, the variability model is connected to various model elements of software development process, by using traceability links. Using a single representation concept to depict the variability of the complete software development process fits in the global picture of the variability management. Therefore, OVM is very useful for global software product line requirements engineering and we adapt the meta-model of OVM (shown in Figure 1) for the managing variability. According to Pohl in [2], the variability information is distributed through out the artifacts of software development (e.g. use cases, classes, test cases etc) and is not localized in the features. Due to this criticism, we are not confined to feature modeling. Further this paper follows the representations of the variability model proposed in [2].

2.2 SYSIPHUS

SYSIPHUS [6] is the global software engineering approach, which gives equal weight to the system models, the communication models and the organizational models. Further it unifies them by deriving these models from a single meta-model as shown in Figure 2. The issue model (a model in Figure 2 with classes *Issue*, *Option*, *Criterion*, *Assessment*, and *Resolution*) of SYSIPHUS is same as the QOC model. Using the *annotates* association, each object of *ModelElement* can be linked to many objects of *Issue* and thus rationale discussions are initiated. Within the rationale discussions options, criteria and assessments are proposed by various stakeholders (located in different places). Using this information resolutions (decisions) are made on the issues. Thus on the basis of rationale discussions, SYSIPHUS uses the issue model as a communication model, and enables informal collaboration in the global software engineering and is evaluated empirically in [6]. The decisions of the rationale discussions are assigned to the objects of *ActionItem,* which are linked with the *OrganizationalUnit.* Following the association between *OrganizationalUnit* and *SystemModel* the action items are performed by a stakeholder in the organizational model. SYSIPHUS supports requirements engineering on the basis of rationale management. The criteria of the issue model can be goals and non-functional requirements. As per the recently published state of SYSIPHUS, it does not support variability management. The process free SYSIPHUS tool is open source software downloadable at [26].

2.3 Rationale-based Variability Management in Requirements Engineering

In the issue model (which is same as QOC) an issue is associated with many options, and an option (or options) is justified using arguments and criteria. In a typical variability management problem, a variation point is associated with many variants and some variants are to be instantiated for a specific product. Therefore issue model is a generic method to manage variability [3]. Rationale-based Variability Management in Requirements Engineering (RVMRE) [3] is our first attempt to use issue model for the variability management. It advocates the combination of the issue model and the OVM for the variability management and proposes:

- *Rationale-based product instantiation* is a technique to instantiate variation points using the variability of issue model.
- *Rationale-based variability constraints* is a technique to identify variability constraints using rationale discussions.

Though RVMRE exploits the variability of issue model, it does not come up with a technique to model variations using the issue model. Kruchten [13] proposed that the rationale elements can be constrained using some dependencies and constraints [13]. We identified that these constraints and dependencies used in the rationale management community are similar to that of variability management. As SYSIPHUS uses the same issue model as a communication model, the variability meta-model can be realized as a special type of the issue model by exploiting its variability (as per RVMRE) and using the constrains of rationale (as proposed by Kruchten). This is the motivation behind issue-based variability modeling and is the contribution of this paper. Using this we address the problems shown in section 1.

3. Related Work

The capture of architectural knowledge in product line engineering is emphasized by [19]. The use of knowledge-based product configuration techniques for product instantiation are proposed by [22] and [23]. These techniques use product context knowledge for the instantiation of variations, but do not address the capture of rationale behind variations and its reuse for the product instantiation and product line evolution. [20] is an ongoing research to support collaboration for variability modeling by sharing the variability models in a distributed team and linking

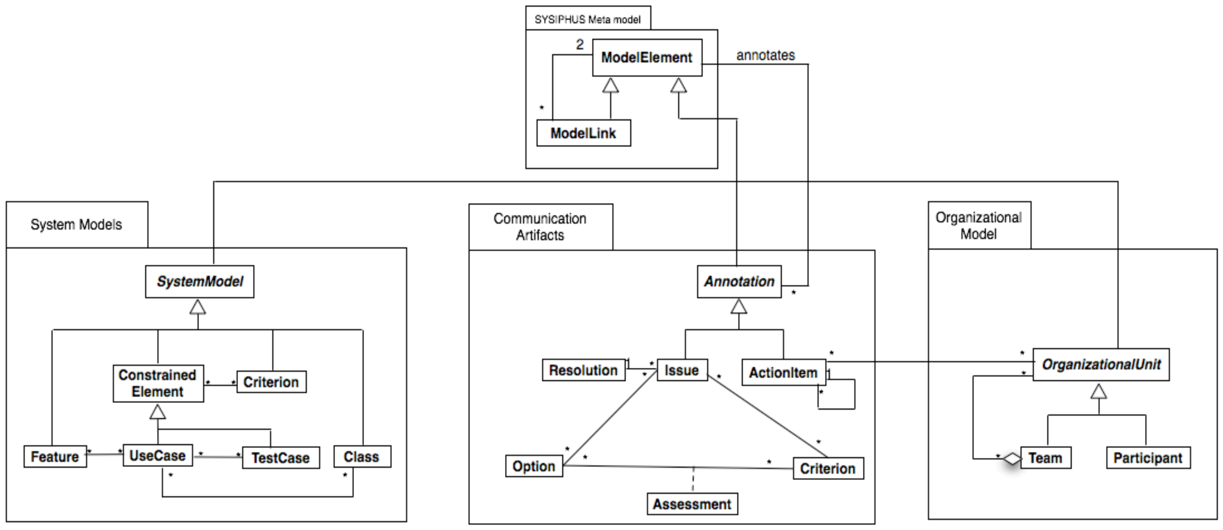


Figure 2: SYSIPHUS, the unified software development approach

variability models to different assets, but it does not address the use of communication models and negotiations for the activities like variability identification, product instantiation and variability evolution. Further [20] attempts to use decisions as variation points, but it does not come up with a variability meta-model using the rationale artifacts. [21] is a way of using collaboration in product line scoping. All the above-mentioned product line approaches do not address the collaboration problem in product line requirements engineering especially in variability identification, product instantiation and variability evolution, and do not address the problems presented in section 1.

The challenges in the distributed requirements engineering are presented in [21]. Requirements negotiation approaches such as EasyWinWin [14] can enable stakeholder collaboration in requirements engineering. The use of asynchronous communication in the context of global software engineering is addressed in [18] and the cost estimation is addressed in [15]. Information models can improve the stakeholder communication in requirements engineering [24]. Variability modeling techniques such as [2], [7] and [8] consider the variability models as an integral part of system models but not as a part of the communication model or the rationale articats. Therefore realizing variability meta-models within the communication artifacts (or the rationale artifacts) is new, and therefore is the innovation of this paper.

4. Infotainment systems

The information in this section is based on an empirical study in an automotive organization, which is planning to move into global product line development in the near future. Infotainment systems are one of their key products and have four major business branches: radios, in-car navigation systems, mobile telecommunication systems and in-car entertainment systems. Each of these business branches has product lines. In-car navigation system has a current business estimate of 20 billion USD with 70% of its current business focused in USA and Japan. In the near future, its business is forecasted to have bright prospect in the booming economies such as India and China. The organization of our consideration has their development centers in India and China, and therefore they are planning to move into the global product line development. To enable this we are researching on the techniques to support variability management in the globally distributed requirements engineering setup, with a major focus on the infotainment application domain.

5. Issue-based variability modeling

The issue model of the SYSIPHUS can be modified to obtain a constrained issue model as shown in the Figure 3, which is the meta-model for the issue-based variability modeling. This meta-model of the issue-based variability modeling is engineered by combining pieces of concepts as follows:

Variation points and variants: According to RVMRE, the *VariationPoint* and *Variant* of the OVM are similar to the *Issue* and *Option* classes of the issue model. In order to exploit this concept, V*ariationPoint* is realized as a subclass of the *Issue* and *Variant* is realized as a subclass of *Option* (refer Figure 3). Doing this we support the modeling of variation points in domain engineering using issues, and enable their instantiation using the justification matrices of the issue model in application engineering.

Constraints in the issue model: According to Kruchten in [13] decisions can be modeled using the relationships such as constrains, forbids, enables, subsumes, conflicts with, is bound to etc. We identified that these relationships are very similar to the constraints of the variability meta-model. Therefore we improvise by introducing the constraints (requires and excludes) between the options of the issue model as well as exploit the same for the variability meta-model. As per the dashed association between the self-association of *Option* and *Constraint Dependency,* options (as well as variants) can be constrained using *Requires* and *Excludes*.

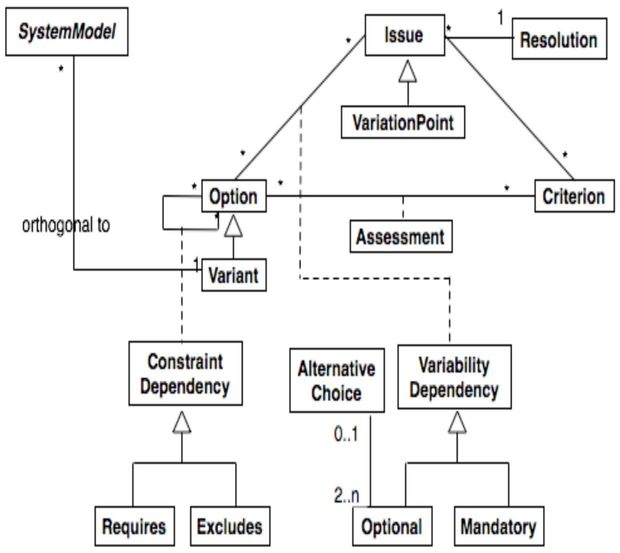


Figure 3: Constrained issue model, a meta-model for issue-based variability modeling

Variability dependencies in the issue model: In our observations, some options (proposed during the rationale discussions) of the issue model can be mandatory i.e they are decisions, some can be optional and some can be optional alternatives. Therefore the association between *Issue* and *Option* can be modeled using a *VariabilityDependency*, which can be *Mandatory* or *Optional*. The *Optional* dependencies can be further constrained using *AlternativeChoice*. Thus the dependencies of the variability meta-model can be introduced into the issue model as well.

Orthogonality of variation points and variants: We propose an *orthogonal to* association between the classes *Variant* and *SystemModel* (of SYSIPHUS). This concept enables us to link the variants of the issue model to all the system model elements (such as features, use cases, classes, test cases etc). This is the orthogonality of the variants. In the existing SYSIPHUS (Figure 2), *ModelElement* can be associated with many instances of *Annotation*, therefore *SystemModel* can be linked to an *Issue*, which is the orthogonality of the variation point. Thus we can use the orthogonality concept of the OVM, to link the system model elements with the communication artifacts, which are variation points and variants.

Thus exploiting the variability of the issue model (as proposed by RVMRE), constraining the issue model and tailing OVM orthogonality for the communication, we engineered the variability meta-model on the issue model (also a communication model), which paves the way for a new concept called issue-based variability modeling. Because of the overlap between the variability meta-model and the communication model, the variation points modeled as issues can be instantiated using the justification matrices enabling informal collaboration of the stakeholders. Issue-based variability modeling uses the graphical representations of variation points, variants, constraints and dependencies as proposed by Pohl in [2]. Figure 4 is a simple illustration from our in-car navigation system case studies, which depicts the variability in the functional requirements. The *Routing Management* variation point has optional variants such as *Voice guidance* and *Automatic accident notification,* and optional alternatives with variants such as *Automatic routing*, *Live traffic data*, *Traffic congestion* and *Rerouting*. The variability model is connected to the use case models using traceability (see Figure 4), i.e variability model is orthogonal to the use case models. Managing the variability in *Routing Management* is the running example of this paper.

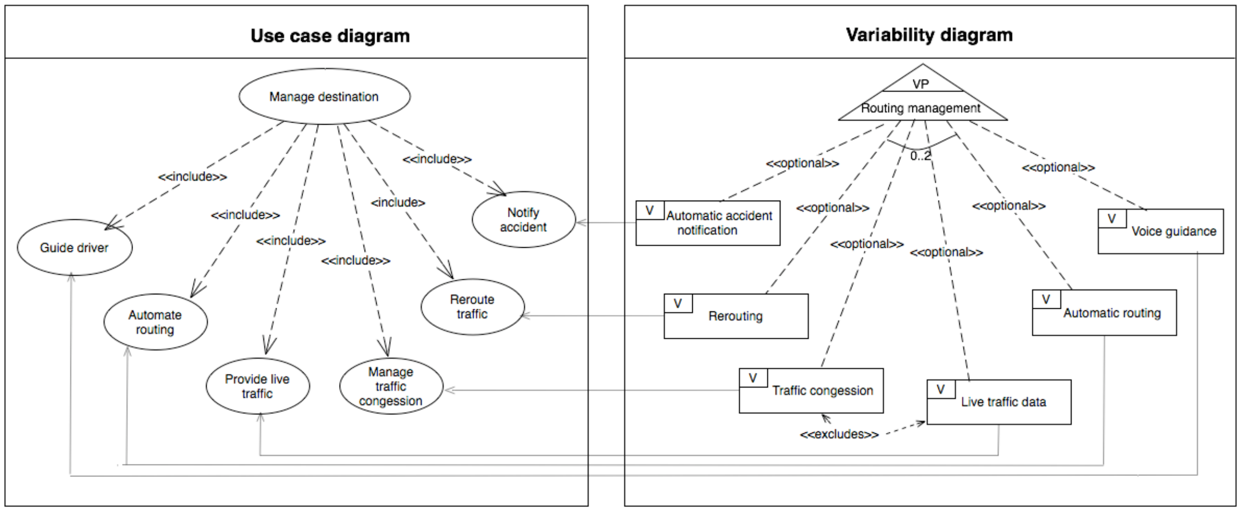


Figure 4: Variability in in-car navigation systems using issue-based variability modeling

6. Variability management

The meta-model of the issue-based variability modeling (refer Figure 3) supports variability modeling similar to OVM [2], additionally it supports the instantiation and evolution of the variation points, capture of rationale as well as supports informal collaboration. This is the big boost of using issue-based variability modeling when compared to the conventional variability modeling techniques such as OVM [2], FODA [8] and Gomma's UML extensions [7]. The important activities of the variability management such as product instantiation, capture of rationale and evolution of variability are presented in the subsections 6.1, 6.2 and 6.3. All these activities are illustrated using the example of Figure 4.

6.1 Product instantiation

The product line customers have their own set of concerns and the product line needs to support the instantiation of the products based on the quality concerns of the customers. Therefore the product instantiation is to be done using the collaboration of domain engineers, application engineers and customers, and is a collaboration intensive problem. Furthermore product lines are tending to support the instantiation of products for global markets. In this case, product line development has domain engineering in a country and the application engineering teams and customers from different countries and cultures of the globe. Therefore there are very important aspects of the globalization and distribution that are to be handled in product instantiation, and the issue based variability modeling attempts to do that. In this specific case of instantiating *Routing Management*, 18 stakeholders (domain engineering team 7, application engineering team 9, customer team 2) with various cultural and academic/professional backgrounds, located in multi-sites are involved. Table 1, shows the justification matrix for the instantiation of the *Routing management*, which is constructed as follows:

- In the view of application engineering, the *Issue* of the variation point is "How to instantiate a variation point?" and the optional variants constitute the options of the justification matrix (refer Table 1).
- The product specific quality concerns of the customer and the product specific goals constitute the criteria of the justification matrices.
- All the stakeholders involved in the product instantiation process give their arguments, i.e fill the cells using the SYSIPHUS tool. SYSIPHUS supports various arguments such as + (supports), ++ (supports strongly), 0 (no effect), - (hinters) and – (hinters strongly).
- Based on the arguments, a resolution is made which supports the variants *Voice guidance*, *Live traffic data* and *Automatic accident notification* for the city car related to the customer XXX.

Thus using the informal collaboration of the stakeholders, issue-based variability modeling supports the instantiation of the variability models in a distributed setup of domain engineering, application engineering and customer teams. Further the simple notations of the justification matrices, is the medium of communication between people of various cultures.

Issue: How to instantiate *Routing Management* for a city car product to a customer XXX?								
Criteria	***Usability***	***Memory***	***Price***	***Traffic intensity***	***Security***	***Reliability***	***Extensibility***	***Maintainability***
Option1: *Voice guidance*	++	*0*	*0*	++	*0*	+	+	+
Option2: *Automatic routing*	++	*0*	--	+	*0*	*0*	*0*	*0*
Option3: *Live traffic data*	++	+	*0*	+	*0*	+	+	+
Option4: *Rerouting*	--	-	-	*0*	*0*	-	*0*	+
Option5: *Automatic accident notification*	*0*	*0*	++	+	++	*0*	+	*0*
Resolution: *Voice guidance*, *Live traffic data* and *Automatic accident notification* are decided to be instantiated for the city car.								

Table 1: Justification matrix for product instantiation, a tabular view of the variability model for the resolution of the variation points using informal collaboration

As per the issue-based variability modeling, the options of the justification matrices are the variants, which are linked to the system model elements (in this case use cases). Therefore following the links from the options to system model elements the instantiated artifacts for a specific product are clear. Please note that variability modeling in domain engineering (Figure 4) is done using the issue-based variability modeling and the instantiation of the variation points in application engineering are done using the justification matrices (justification matrix is a tabular view of the variability models). This is the major improvement brought by our technique when compared to conventional variability modeling techniques.

6.2 Variability identification and capture of rationale

The expertise of domain engineering is often distributed. Variability identification may involve the collaboration of stakeholders from application engineering as well. Therefore variability identification is also a collaboration intensive problem. In this section we show a way of identifying the dependencies and constraints of the variability model on the basis of the brainstorming and the informal collaboration provided by issue model. Here the rationale behind the variability models is captured within the process of the variability identification. This is done on the basis of the issue model of SYSIPHUS by using the following abstractions:

Issue triggers the identification of a variability dependency or a constraint by posing questions on it. For e.g. issue of the Table 2 initiates the identification of the variability dependencies.

Option supports/hinters a possible variability dependency or constraint that corresponds to an issue (for e.g. please refer options of Table 2).

Criterion is the representation of the mass-customization forces that cause variations. Mass-customization forces can be non-functional requirements (NFRs) and goals that come from business, marketing, technology, project management and legal issues. For e.g. the criteria that cause the variability dependency of *Voice guidance* are *Usability*, *Memory* and *Price* which are also presented in Table 2.

Issue: What is the variability dependency of the *Voice Guidance* in *Routing Management?*			
Criteria	***Usability^{PL}***	***Memory^{PL}***	***Price^{PL}***
Option1: *Mandatory* dependency is supported	++	--	0
Option2: *Optional* dependency is supported	0	+	+
Resolution: Variability dependency of *Voice guidance* is decided to be optional because of better assessments in price and memory.			

Table2: Justification matrix for variability dependency

In the case of the justification matrix of Table 2, variability identification is performed using the collaboration of 5 stakeholders, sitting in different locations. Further Table 2, is the rationale behind the mandatory dependency and therefore the variability rationale (rationale behind the variations) is captured within the process of variability identification, and provides reasoning and justification behind the variations. Similarly we can identify the constraint dependencies (requires & excludes e.g. Table 4) and alternative choice (Table 3), as well as capture the rationale behind them using the involvement of distributed expertise.

Issue: What is the alternative choice of the *Routing Management?*				
Criteria	*Usability*^{PL}	*Memory*^{PL}	*Price*^{PL}	*Traffic intensity*^{PL}
Option1: A range of 0..2 is supported.	++	0	0	+
Option2: A range of 0..3 is supported.	--	--	--	0
Resolution: Option1 is decided to be alternative choice of the *Routing Management*, because of better usability and support of traffic intensity.				

Table3: Justification Matrix for Alternative Choice

Issue: What is the constrain dependency between *Live traffic data* and *Traffic congestion?*			
Criteria	*Usability*^{PL}	*Memory*^{PL}	*Price*^{PL}
Option1: *Traffic congestion requires Live traffic data*	++	0	--
Option2: Bi-directional *excludes* between Live traffic data and Traffic congestion	0	+	+
Option3: No constraint dependency is required	0	0	--
Resolution: Option2 is decided because of better assessments in memory and price.			

Table4: Justification Matrix for Constraint Dependency

In section 5, we model variation points as issues. In the view of domain engineering, the issue of the variation point is "Why do we have variation?". In SYSIPHUS, an issue can be linked to many issues (refer Figure 1). So the variation point is linked to the issues of variability dependencies and constraints (in the running example Table 2, Table 3 and Table 4). These justification matrices answer the issue of the variation point and further address the Problem1 of Section1. Thus, linking the issue of the variation point to the issues of dependencies and constraints, captured rationale information is integrated into the variability models. Doing this we provide model-based reasoning and justification for variations along with the representation of mass-customization forces in the form of criteria. Further the criteria of the justification matrices captured in the variability identification can be reused for product instantiations, e.g. criteria *Memory*, *Usability, Traffic intensity* and *Price* are reused from Table 2, Table 3 and Table 4 to Table 1. Please note that, while reusing the criteria of the captured variability rationale for the product instantiation, their qualities are to be adjusted based on the quality concerns of the customers. This is done using the conflict resolution technique of SYSIPHUS.

Capturing rationale for variations improves the global understandability of the variability models. Further rationale provides information related to the instantiation of the variability models (e.g. criteria of the variability rationale). As the variability models are to be instantiated for global markets, variability models with rationale as per issue-based variability modeling are very useful in the context of global product line engineering.

Issue: What is the variability dependency of the *Voice Guidance* in *Routing Management?*			
Criteria	*Usability*^{PL}	*Memory_1*^{PL}	*Price_1*^{PL}
Option1: *Mandatory* dependency is supported	++	+	0
Option2: *Optional* dependency is supported	0	-	0
Resolution: Variability dependency of *Voice guidance* is decided to be mandatory because of better assessments in usability and memory.			

Table5: Update of justification matrix

6.3 Variability evolution

The capture of the variability rationale can support the evolution of the variability models. As per Table 2, *Voice guidance* is justified to be optional. On the passage of time *Memory*^{PL} and *Price*^{PL} criteria of Table 2 are changed to *Memory_1*^{PL} and *Price_1*^{PL}. Due to the change of the mass customization forces, the state of the justification matrices has changed (from Table 2 to Table 5) and this leads to turning *Voice guidance* into a mandatory variant. In this particular update is done using the collaboration of two stakeholders, who are not involved in the building Table 2. Please note that the product lines live for a longer period of time, and the capture of variability rationale can support their evolution continuously over their life.

6. Empirical evaluations

Issue-based variability modeling enables the modeling of variation points using issues in domain engineering and support their instantiation using justification matrices (a specific view of the issue model) in application engineering. Therefore domain engineering and application engineering are done on the same basis i.e. issue model. Further the criteria of the variability rationale captured in domain engineering, is reused in application engineering. This can bring domain engineering and application engineering together and addresses Problem2. Activities like variability identification (Table 2, Table 3 and Table 4), product instantiation (Table 1) and variability evolution (Table 5) are done using the informal collaboration of the stakeholders. This addresses Problem3. The capture of the rationale behind variations addresses Problem1 as discussed in section 7. Thus issue-based variability modeling addresses the problems raised by this paper.

The issue-based variability modeling is implemented as the variability plug-in to the SYSIPHUS by an industrial organization. This tailored most of the SYSIPHUS functionalities such as traceability, justification matrices etc for the product lines and provides extensive tool support for global product line requirements engineering. Please note that the existing state of SYSIPHSUS as published in [6] supports sharing models in a distributed environment and enables the distributed design and update of models based an event mechanism. The variability-plugin of SYSIPHUS exploits the same techniques to share the justification matrices and the variability models. Using the variability plugin an experiment is conducted with in a group of 23 people (not related the automobile company of section 4) with various cultural and professional backgrounds. The people are randomly divided into an experimental group of 12 (using SYSIPHUS with issue-based variability modeling) and a control group of 11 (using OVM and communicating on Skype, telephones and emails). Both the experimental group and the control group members are distributed in different locations, in a similar way. This experiment considered 53 variation points from the domains of domestic appliances and infotainment systems obtained from the industrial requirements specifications. Both the experimental and control groups worked on the same variation points. In this we observed that:

- The average time taken to instantiate variation points in the experimental group was lower (by 30%) to that of control group.
- We performed several product instantiations, and on average 76% of the criteria of the captured variability rationale is reused in the experimental group. This justifies the capture of variability rationale as per section 7. Please note that the reuse of criteria is illustrated in section 5 (criteria from Tables 2, 3 & 4 to Table 1).
- At the end of the experiment a self-administrated questionnaire was given to the participants of the experimental and the control group. Q1 (please refer appendix section) is a research question in the questionnaire given to the participants. The responses were collected from them. The response choices (*Very high*, *High*, *Fair*, *Low*, *Very low* and *No information available*) are coded [25], and the mean quality is computed for the response frequencies of both the experimental and control group data. The mean quality of the data (from Q1) collected from the experimental group is 80% more than the mean quality of the control group data. This justifies that the issue-based variability modeling provides more information to change variations than the OVM.

We did an attempt to use a descriptive survey for evaluating the application of the notations of the justification matrices for global requirements engineering, with interview as a survey instrument [25]. The idea of this survey is to measure the qualities of the simplicity related parameters of the justification matrices such as *understandability*, *easiness*, *learnability*, *usefulness for informal collaboration* and *willingness to adopt* in cultures such as European, Chinese and Indian using the research questions Q2, Q3, Q4, Q5 and Q6. 20 professionals are sampled from each culture and are interviewed. We observed that all the parameters recorded a mean quality higher than 93% in all cultures. This gives some evidence that the representations of the justification matrices are understandable in various cultures and therefore can be a good medium of communication between various cultures. Further variability rationale represented using these justification matrices (which are understood in various cultures) improves the understandability of the variability models in various cultures.

7. Conclusion

SYSIPHUS is the global software engineering technique, which uses the issue model (a rationale model) as the communication model. This paper proposes a new technique to model variations, issue-based variability modeling, which is obtained by engineering the variability meta-model on the issue model of SYSIPHUS. Issue-based variability modeling is implemented as a variability plug-in to the tool SYSIPHUS and is evaluated empirically using an experiment and a survey. From our research, implementation and empirical evaluation experience we conclude that:

- Issue-based variability modeling supports variability modeling similar to the existing techniques (such as OVM & FODA), and uses communication artifacts of SYSIPHUS to model variation points. Additionally it supports aspects such as informal collaboration, capture of rationale, instantiation and evolution of variation points. Further the issue-based variability modeling uses the graphical representation standard proposed by Pohl in [2].
- The notations of the justification matrices are very simple notations, which can be understood in various cultures. Therefore they are good representations of the communication artifacts. This is also evaluated using a survey.
- The empirical evaluations of this paper show positive results on the aspects like instantiation support and evolution support of the variability models. But they do not validate aspects like external validity, usability in a various teams with different skill and motivation levels, and the long-term usefulness of rationale. Therefore they are planned for future.

Acknowledgements

We would like to thank Dr. Peter Amthor (Siemens CT SE1), Dr. Allan Dutoit (TUM) and Dr. Guenter Boeckle (Siemens CT SE3) for their support in our research.

References

[1] P. Clements and L. Northrop, “A Framework for Software Product Line Practice-Version 4.2 [online]”. *Carnegie Mellon, Software Engineering Institute* URL: http://www.sei.cmu.edu/prodvolnuctlines/framework.html, Pittsburgh, USA, 2006.

[2] K. Pohl, G. Böckle, F. van der Linder, *Software Product Line Engineering Foundations, Principles, and Techniques,* Springer 2005.

[3] A.K Thurimella, “Rationale-based Variability Management in Product Line Requirements Engineering”, IASTED *Software Engineering 2007,* Innsbruck, February 13 to 15 2007.

[4] A. Dutoit, R. McCall, I. Mistrik, B. Paech, *Rationale Management in Software Engineering,* Springer 2006.

[5] Allan MacLean, Richard M. Young, Victoria M.E. Bellotti, and Thomas P. Moran, “Questions,options, and criteria”. *Elements of design space analysis*, Human-Computer Interaction, 6(3-4),1991, 201–250.

[6] B. Bruegge, A.H. Dutoit, T. Wolf, “Sysiphus: Enabling informal collaboration in global software development. In Proceedings of the First International Conference on *Global Software Engineering”*, Costão do Santinho, Florianópolis, Brazil, October 16-19 2006.

[7] H. Gomaa, *Designing Software Product Lines with Uml: from Use Cases to Pattern-Based Software Architectures*, Addison Wesley Longman Publishing Co., Inc 2005.

[8] Kang.C.Kyo, Sholom, G.Cohen, James, A.Hess, William, E.Novak, and A.Spencer Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Study”, Technical Report: CMU/SEI-90-TR-21. *Software Engineering Institute. Carnegie Mellon University,* 1990.

[9] L. Hotz, T. Krebs and K. Wolter, “Using a Structure-based Configuration Tool for Product Derivation”, *19th IEEE International Conference on Automated Software Engineering (ASE'04)*, 2004, pp. 388-391.

[10] Anton Jansen, Rein Smedinga, Jilles van Gurp and Jan Bosch, “Feature-based Product Derivation: Composing Features”, *IEE Proceedings Software - special issue on Software Engineering*, August 2004.

[11] Sybren Deelstra, Marco Sinnema and Jan Bosch. Product Derivation in Software Product Families: A Case Study, *Journal of Systems and Software*, Volume 74, Issue 2, 15 January 2005, pp. 173-194.

[12] T. Krebs, K. Wolter and L. Hotz, Model-based Configuration Support for Product Derivation in Software Product Families. *Workshop Planen, Scheduling und Konfigurieren, Entwerfen (PuK2005)* -- KI 2005 Workshop

[13] P. Kruchten, "An ontology of architectural design decisions in software intensive systems". In 2nd *Groningen Workshop on Software Variability*, December 2004, pages 54--61.

[14] P. Grünbacher, “Collaborative Requirements Negotiation with EasyWinWin”, 2nd *International Workshop on the Requirements Engineering Process*, Greenwich, London, IEEE Computer Society, 2000, pp. 954-960.

[15] P. Keil, D. Paulish, R. Sangwan, “Cost Estimation for Global Software Development”, *Proc. of the Int. Workshop on Economics-Driven Software Engineering Research (EDSER), Shanghai, China, 2006,* pp. 7-10.

[16] Izquierdo, L., D. Damian, and D. German, “Requirements management in special case of global software development: a study of asynchronous communication in the open source community”, *Proc. of*

Int. Workshop on Distributed Software Engineering, Paris, Aug. 2005
[17] Damian, D. and Zowghi, D., “Requirements Engineering challenges in multi-site software development organizations”, *Requirements Engineering Journal,* 8, 2003, pp.149-160.
[18] D. Herlea and S. Greenberg, “Using a groupware space for distributed requirements engineering”, Proc. of the Seventh *IEEE International Workshop on Enabling Technologies*: Infrastructure for Collaborative Enterprises, June 1998, pp. 57-62.
[19] Deepak Dhungana, Rick Rabiser, Paul Grünbacher, Christian Federspiel, Klaus Lehner, "Architectural Knowledge in Product Line Engineering: An Industrial Case Study", 32nd Euromicro Conference on *Software Engineering and Advanced Applications (SEAA)*, Cavtat/Dubrovnik (Croatia), August 29-September 1, 2006
[20] Deepak Dhungana, Rick Rabiser, Paul Grünbacher, "Coordinating Multi-Team Variability Modeling in Product Line Engineering", Proceedings *2nd Workshop on Supporting Knowledge Collaboration in Software Development*, Tokyo, Japan, 2006.
[21] Muhammad Asim Noor, Rick Rabiser, Paul Grünbacher, "A Collaborative Approach for Reengineering-based Product Line Scoping", Proceedings *1st International Workshop on Agile Product Line Engineering (APLE'06)*, Baltimore, USA, 2006
[22] Rick Rabiser, Deepak Dhungana, Paul Grünbacher, "Integrating Knowledge-Based Product Configuration and Product Line Engineering: An Industrial Example", ECAI 2006 *Workshop on Configuration, Affiliated with the 17th European Conference on Artificial Intelligence (ECAI 2006)*, August 28 - 29, Riva del Garda, Italy, 2006
[23] L. Hotz, A. Gunter, and T. Krebs, “A Knowledge-based product derivation process and some ideas how to integrate product development”, In Proc. of *Software Variability Management Workshop*, pp. 136--140, Groningen, The Netherlands, (February 13-14 2003).
[24] Paech, B., Dorr, J., and Koehler, M. 2005, “Improving Requirements Engineering Communication in Multiproject Environments”, *IEEE Softw,* volume 22, Jan. 2005, pp 40-47.
[25] A. Fink, *The survey handbook*, Sage Publications Inc, 2003.
[26] sysiphus.in.tum.de

Appendix: Research questions

Q1. Suppose you have change requests for the variations in the variability model. Is there useful information in the variability model that helps you to justify, if the change requests can be applied?

Answer: Please select only one of the following

[ ] Very high
[ ] High
[ ] Fair
[ ] Low
[ ] Very low
[ ] No information available
[ ] Don’t know the answer

Additional Comments:

Q2. To what extent do you understand the representations of the justification matrices?

Answer: Please select only one of the following

[ ] 90 to100% (1)
[ ] 70 to 90 % (2)
[ ] 40 to 70% (3)
[ ] Below 40% (4)
[ ] Not at all (5)

Additional Comments:

Q3. How easy/difficult for you are the representations of the justification matrices?

Answer: Please select only one of the following

[ ] Very easy
[ ] Easy
[ ] Fair
[ ] Difficult
[ ] Very difficult
[ ] Don’t know

Additional Comments:

Q4. How easy/difficult is for you to learn the representations of the justification matrices?

Answer: Please select only one of the following

[ ] Very easy
[ ] Easy
[ ] Fair
[ ] Difficult
[ ] Very difficult
[ ] Don’t know

Additional Comments:

Q5. How useful are the representations of the justification matrices for the informal collaboration between people of various cultures?

Answer: Please select only one of the following
[ ] Very useful
[ ] Fairly useful
[ ] Useful

[ ] Less useless
[ ] Useless
[ ] Don’t know

Additional Comments:

Q6. Do you adopt the representations of the justification matrices for the informal collaboration? Why?

Answer: Please select only one of the following

[ ] Definitely
[ ] Probably
[ ] Maybe
[ ] No
[ ] Don’t know

Additional Comments:

The Effects of Communication Mode on Distributed Requirements Negotiations

Teresa Mallardo[1], Fabio Calefato[1], Filippo Lanubile[1], Daniela Damian[2]
[1]University of Bari, Dipartimento di Informatica, Bari, Italy
[2]University of Victoria, Department of Computer Science, Victoria, Canada

Abstract

Videoconferencing is generally considered as the most appropriate medium to conduct requirements negotiations between remote stakeholders. To improve the effectiveness of distributed requirements negotiations, drawing upon the postulates of theories on media selection, we argue that a combination of lean and rich media is needed. In this paper we empirically test the hypothesis that the early resolution of uncertainties through an asynchronous lean medium can shorten the list of open issues to be negotiated over a synchronous rich channel.

1. Introduction

Requirements negotiation is one of the most complex and communication intensive practice of software engineering, especially in distributed scenarios, where arranging collocated meetings is often impractical. Previous studies in the field of Requirement Engineering [5] [9] indicate that videoconferencing is the most appropriate medium for effectively conducting distributed negotiations, thanks to its synchronicity (i.e., the capability of conveying information in a timely manner) and richness (i.e., the ability to convey the sense of physical presence of individuals, as well as a number of visual and verbal cues). However, while videoconferencing sessions come with an additional overhead (e.g., the costs of infrastructure setup and maintenance), even when everything runs smoothly [11], it is still hard to conduct a long-running and productive discussion during a videoconference, especially when more than a few people are involved. In contrast, asynchronous lean media, such as email or discussion forums, lacks all these abilities (e.g., one cannot see people nodding in text-based communication). Thus, to improve the effectiveness of distributed requirements negotiations, drawing upon the postulates of theories on media selection, we argue that a combination of rich synchronous media and lean asynchronous media is needed.

The Media Richness theory [1] [2] is one of the most prominent in the field of computer-mediated communication (CMC) studies. It posits the existence of two complementary forces, namely uncertainty, which act on individuals when they process the information exchanged to execute a task. *Uncertainty* represents the lack of required information, whereas *equivocality* represents the existence of multiple and conflicting interpretations of available information However, during the execution of a complex task like requirements negotiations, communicating and agreeing on requirements involves a constant interplay between both collecting further information about requirements and their context (i.e., uncertainty reduction), and resolving ambiguities, misunderstandings, or conflicts in requirements (i.e., equivocality reduction) [10]. In addition, the Media Switching theory [12], a more recent theory on CMC, has analyzed communication from a cognitive perspective, arguing that while rich media are useful in ensuring commitment to the task execution, they allow individuals a substantially lower ability to properly (re)process information at will, as compare to lean media. Thus, from the consistent combination of these two theories we argue that, on the one hand, rich synchronous communication is better suited for resolving the ambiguities that may arise in the discussion of requirements issues. On the other hand, when discussing issues or inspecting requirements documents, stakeholders may also need time to process information properly and sift through the issues outside of the meeting, at will and in a less interactive manner. Hence, lean asynchronous communication can more effectively support stakeholders in thoroughly analyzing issues, as well as in resolving issues of uncertainty by conveying missing information.

In two of our previous studies [3] [4], we have already shown that asynchronous discussions improve the effectiveness of synchronous requirements negotiations. Instead, in this paper we aim at investigating the hypothesis that the resolution of uncertainties through an asynchronous discussion, conducted before the synchronous negotiation meeting, can shorten the list of requirements with open issues to be negotiated in a real-time manner. Rich media negotiation meetings will thus be mostly focused on reducing ambiguities (equivocality) in requirements. In this way the overall effectiveness of the requirements engineering process can be increased by cutting down the number of issues that remain open after the final synchronous negotiation.

The remainder of the paper is organized as follows. Section 2 describes the experiment in detail, including the design, the variables and hypotheses, and the threats to validity. Section 3 describes the results whereas Section 4 discusses the findings from the experiment. Finally, conclusions and future work are presented in Section 5.

2. The Empirical Study

The study was performed during a software engineering course, held in Spring 2005, and organized by three universities: University of Bari (Italy), University of Victoria (Canada), and University of Technology, Sydney (Australia).

Thirty-two students (10 Italians, 12 Canadians, and 10 Australians) were divided into six international project teams. Each team was formed by a client group and a developer group, interacting remotely. All the members of each group were, instead, always collocated. As shown in Table 1, each Canadian and Australian group was involved in two different projects, playing the role of client (C) and developer (D), respectively. Instead, each of the two Italian groups was involved in only one project, either as a client (Gr6cl) or as a developer (Gr6dev).

The study used three distinct projects, each with two instances. Project A (A1 and A2 in Table 1) was to design a Global software development system to facilitate GSD collaboration. In project B (B1 and B2) the students designed the interface for a "iMedia" software to allow users to purchase movies online, organize and play their movies. Finally, project C (C1 and C2) involved the design of a real estate system.

The outcome of each project was a software requirements specification (SRS) resulting from the mutual agreement reached by the client group and the developer group. This mutual agreement was developed through a series of scheduled activities. First, a Request for Proposal (RFP) was produced by the client group and discussed during the requirements elicitation meeting, held in a videoconference by the entire team (both clients and developers). Then, the SRS was developed by the developer group in each project, with the client team providing feedback. This feedback had been provided earlier, through an inspection entirely performed online with the help of the IBIS tool [8]. The inspection was carried out individually by each member of the client team, who participated in the Discovery stage by reading the SRS and recording issues in the system. Each recorded issue was classified according to the IEEE standard taxonomy for good requirements documents (i.e., as omission, ambiguous info, incorrect fact, inconsistent info, not verifiable, or not modifiable) [7]. One of the researchers collected all issues and merged duplicates (i.e., issues found by more than one client) into a unique list of collated issues.

Table 1. Groups of clients (C) and developers (D) allocated to course projects

Country	Group	Project A (A1, A2)		Project B (B1, B2)		Project C (C1, C2)	
		PT1	PT2	PT3	PT4	PT5	PT6
Ca	Gr1	C					D
	Gr2		D	C			
	Gr3				D	C	
Au	Gr4	D			C		
	Gr5		C			D	
It	Gr6cl						C
	Gr6dev			D			

After the inspection, three teams out of six participated in a four-day asynchronous discussion using IBIS (i.e., in the Discrimination stage), and the other three teams jumped into the negotiation without asynchronous discussion. The purpose of the asynchronous discussion was to reach an understanding of each issue and identify those issues that could be closed online (i.e., where resolution could be reached without further negotiation) or remained open issues (everything else, which had to be further negotiated in real-time discussion). The process of closing issues used two mechanisms in IBIS: a discussion thread consisting of messages with respect to a certain issue was created, and voting as to whether it is still an open issue or is resolved and thus could be closed.

Finally, all six teams attended the requirements negotiation, which was held in a one-hour videoconference meeting session involving the remote developers and clients. The three teams that asynchronously discussed prior to the negotiation had to resolve only those issues that could not be closed during the asynchronous discussion and thus, remained open issues. The other three teams entered the negotiation with the entire list of issues collated from the inspection.

2.1. Study Design

As shown in Table 2, we manipulated as independent variable the *communication mode*, with the following two treatments: (1) *mixed media* and (2) *rich media-only*.

Clients and developers in the *mixed media* teams used the IBIS tool to asynchronously discuss and store threaded discussions on requirements issues. The aim was to come to an understanding of each issue by exchanging messages and to reach an early resolution through a common agreement expressed by voting. Those open issues that could not be closed during asynchronous discussion were then left for the synchronous requirements negotiation.

Rich media-only teams skipped the asynchronous discussion and all issues found at the discovery stage were thus considered as open issues to be dealt at the negotiation.

Table 2. Study design

project team (client/developer)	communication mode
A1 (gr1/gr4)	rich media-only
B1 (gr2/gr6dev)	rich media-only
C1 (gr3/gr5)	rich media-only
A2 (gr5/gr2)	mixed media
B2 (gr4/gr3)	mixed media
C2 (gr6cl/gr1)	mixed media

2.2. Variables and Hypotheses

To conceptualize the elements in our research hypothesis, we defined the construct of the type of issues being discussed during the asynchronous and synchronous discussions. Our intention was to distinguish between elements of uncertainty and equivocality in the conversations. When an issue indicated the absence of sufficient information in a specific requirement and thus, implied a request of explanation in form of *extra information*, it was classified as *uncertainty*. Conversely, when an issue indicated multiple and possibly conflicting interpretations of a specific requirement and thus, implied a request of explanation in form of *clarification*, with no additional information, it was classified as *ambiguity* (or *equivocality*). Therefore, we measured the number of uncertainties and ambiguities in all asynchronous and synchronous discussions.

To count uncertainties and ambiguities, we parsed all the issues identified during the IBIS-based discovery stage performed by clients. We included in the *uncertainty set* all the issues classified under the category "omission" of the IEEE taxonomy. Similarly, we included in the *ambiguity set* all the issues classified under the category "ambiguous info" of the IEEE taxonomy. The issues classified in the remaining categories of "incorrect fact", "inconsistent info", "not verifiable" and "not modifiable" were also analyzed and counted as part of the one of the two sets depending on whether they required additional information (i.e., could be resolved by removing uncertainty and thus, classified in the uncertainty set) or clarifications (i.e., meaning was ambiguous and had to be clarified and thus, classified in the ambiguity set).

Thus, we formulated the following two hypotheses:

H_1 *During asynchronous discussions of mixed media teams the percentages of closed uncertainties are higher than the percentages of closed ambiguities.*

H_2 *During synchronous negotiations of all teams the percentages of closed ambiguities are higher than the percentages of closed uncertainties.*

To investigate the H_1 and H_2 hypotheses, we collected the following dependent variables:

% closed uncertainties during async discussion = the ratio of closed uncertainties after async discussion to uncertainties after discovery.

% closed ambiguities during async discussion = the ratio of closed ambiguities after async discussion to ambiguities after discovery.

% closed uncertainties during sync negotiation = the ratio of closed uncertainties after sync negotiation to uncertainties before sync negotiation.

% closed ambiguities during sync negotiation = the ratio of closed ambiguities after sync negotiation to ambiguities before sync negotiation.

Where *closed issues* (uncertainties or ambiguities) are issues for which a consensus was reached between developers and clients during discussions, either asynchronous or synchronous.

Furthermore, to investigate the presence of *extra info* and *clarifications* related to issues in the conversation, we performed the content analysis (or coding) on the transcripts of the video recorded synchronous negotiations. One of the researchers identified thematic units[1] within negotiations' transcripts, then two coders performed the coding separately, and finally we counted the number of thematic units classified as *extra info* and *clarifications*. An *extra info* is a category specific for issues classified as uncertainties which raises new information about the issue that has not been elicited yet. A *clarification* is a category for issues classified both as uncertainties and ambiguities which states explanation without adding new information about the issue. Both categories do not include any form of agreement or disagreement expression.

According to the previous hypothesis (H_2), during synchronous negotiations, mixed media teams were more focused on closing ambiguities. Thus, we expected that they provided more clarifications than rich media-only teams. Conversely, because mixed media teams closed most of the uncertainties asynchronously (H_1), during synchronous negotiations they were expected to provide less extra info than rich media-only teams. Therefore, we formulated the following other two hypotheses:

H_3 *Mixed media teams use fewer clarifications than rich media-only teams to reach a consensus.*

H_4 *Mixed media teams use fewer extra info than rich media-only teams to reach a consensus.*

2.3. Threats to Validity

One of the key issues in experimentation is evaluating the validity of results [13]. Thus in the following we report the threats that are relevant for our study.

Threats to *internal validity* influence the conclusions about a possible causal relationship between the treatment and the outcome of a study. The following rival explanations for the findings have been identified. Because in this study there were three different project topics, we cannot exclude that the topic and project complexity could have been a confounding factor. Another threat to internal validity occurs because we were not able to completely randomize the selection and participants' assignment to the different groups. Indeed, while Australian and Canadian students were exposed to both levels of the independent variable, although with different roles (clients or developers), Italian students were not able to work on two projects and had the chance to choose the experimental treatment.

External validity describes the study representativeness and the ability to generalize the results outside the scope of the study. We identified the following threats to external validity. Involving students as subjects of the study (both as clients and as developers) may not be representative of the population of professional stakeholders. However, this threat is partially mitigated by the presence of Canadian students, who were attending a specific course on global software development and then were trained on meeting protocols and negotiation techniques for requirements engineering. Some students had also previous working experience in the software business.

Finally, *conclusion validity* concerns the relation between the treatments and the outcome of the experiment, regarding statistical methods, reliability of measures and treatment implementation. In our study an issue that could affect the statistical validity is the size of the sample data (6 projects, 32 subjects), and for this reason we performed non-parametric tests.

3. Results

To validate the H_{1-4} hypotheses we performed the Wilcoxon matched pairs test as a nonparametric alternative for dependent samples. The Wilcoxon's matched pairs test only assumes that the variables to be compared are on an ordinal scale and that the differences between the two variables can be rank ordered too [13].

In testing H_1, we compared the percentages of closed uncertainties to that of closed ambiguities during the asynchronous discussion for the three mixed media teams. In testing H_2, we compared the percentages of closed uncertainties to that of closed ambiguities during the synchronous discussion for all teams.

With regard to the H_1 hypothesis, Figure 1 shows that asynchronous discussions were more useful to close uncertainties than ambiguities, as expected. Although participants had a high number of uncertainties to be discussed during the asynchronous discussion, they were able to close many of them. During the asynchronous discussions of all the three mixed media teams, the percentages of closed uncertainties (0.53%, 0.91%, and 0.53%, respectively for A2, B2 and C2) were always higher than the percentages of closed ambiguities (0.33%, 0.82%, and 0.0%, respectively for A2, B2 and C2). The Wilcoxon test was significant at the 10% level (Z=1.603, p=0.10).

With regard to the H_2 hypothesis, Figure 2 shows higher percentages of closed ambiguities than closed uncertainties during synchronous negotiation for each

[1] A single thought unit or idea unit that conveys a single item of information extracted from a segment of content [6].

of the six projects, according to our expectation. Also in this case the difference is statistically significant at the 10% level (Z=1.603, p=0.10).

With regard to the H_3 and H_4 hypotheses, we performed the content analysis on the negotiations' transcripts. The inter-coder agreement between the two coders was measured by Cohen's kappa and ranged from 0.84 (for project A2) to 0.94 (for project A1). Our interest was in observing any differences between the numbers of *extra info* and *clarifications* recorded for the rich media-only vs. mixed media teams. In testing the H_3 and H_4 hypotheses we found the following results (see Table 3):

(1) the mixed media teams had significantly higher numbers of clarifications per issue (Z=1.963, p=0.04) than the rich media-only teams;

(2) the number of extra info per uncertainty were significantly lower for the mixed media teams (Z=1.963, p=0.04).

4. Discussion

The quantitative analysis of data indicates that, as compared to the synchronous discussions, in the asynchronous discussions participants closed more uncertainties than ambiguities. Consequently, participants who had already run an asynchronous discussion (i.e., belonging to mixed media teams) could start the videoconference negotiation meeting with a shorter list of open issues to be discussed (mostly ambiguities). Instead, for rich media-only teams more ambiguities than uncertainties were closed during the videoconference negotiation meeting (i.e., the only media participants used).

Moreover, results of the content analysis indicate that a lower number of extra info units were recorded consistently for the mixed media teams. In other words, participants of mixed media teams in the negotiations did not provide additional information for those uncertainties already discussed asynchronously but that remained still open.

Our findings are consistent with the predictions of media selection theories described [1] [2] [12], since asynchronous discussions resulted more effective for reducing the uncertainty in requirements, whereas synchronous discussions more effectively reduced the ambiguity in requirements. In particular, while rich media high in social presence – such as synchronous videoconference meetings – are needed for converging to a shared agreement, lean media low in social presence – such as asynchronous text-based discussions – are valuable in providing an early mechanism to structure the discussion of requirements issues before synchronous negotiation sessions.

Although synchronous videoconferencing meetings ensure project stakeholders' motivation and attention in the discussion of possibly conflicting requirements, the high social presence, important in supporting the social relationships, may also impede unbiased or prompt decisions. Asynchronous text-based

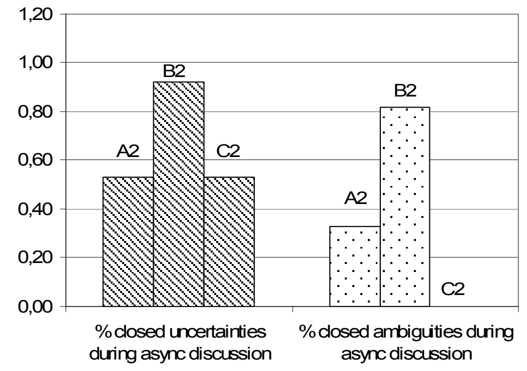


Figure 1. Uncertainty and equivocality reduction during async discussion

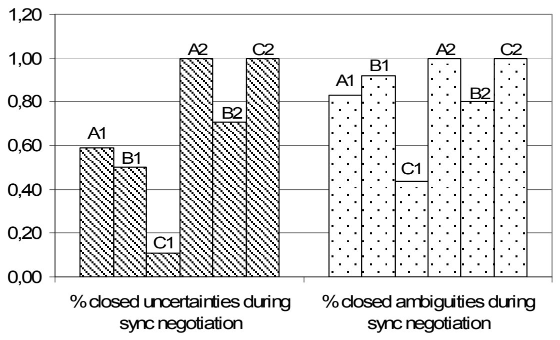


Figure 2. Uncertainty and equivocality reduction during sync negotiation

Table 3. Results from the content analysis

	rich media-only			mixed media		
	A1	B1	C1	A2	B2	C2
discussed issues	34	50	31	12	12	13
thematic units	350	245	298	174	141	125
clarifications per issue †	3.03	1.66	3.32	**5.42**	**4.67**	**3.62**
extra info per uncertainty ‡	1.81	1.09	2.00	**0.63**	**0.86**	**0.44**

† values we compared to test the H_3 hypothesis
‡ values we compared to test the H_4 hypothesis

communication medium emerges as a useful complement in preparation for such meetings: they allow the group participants to process information and consider requirements issues and provide missing information (reducing uncertainty) at their own time and pace. Moreover, asynchronous discussions allowed shortening the duration of synchronous negotiations that were effectively carried out in a one-hour videoconference session.

5. Conclusions & Future Work

In this paper we have presented an empirical study on the effects of rich-media synchronous communication (i.e., through videoconferencing) and lean-media asynchronous communication (i.e., through a web-based discussion forum) in distributed requirements negotiations. The study was conducted in collaboration of three universities in three countries (Australia, Canada, and Italy).

Our findings have shown that, during rich synchronous discussions, remote stakeholders closed a statistically significant higher number of ambiguities than uncertainties. Conversely, during lean asynchronous discussions, stakeholders were able to close a significantly higher number of uncertainties than ambiguities.

These results have a practical impact in the design of a new toolset, which has to include a combination of synchronous/asynchronous media for effectively supporting distributed requirements negotiations. Then, such toolset would be capable of shortening the duration of a synchronous negotiation, conducted over a rich-medium, by running first an asynchronous discussion over a lean medium to cut down the number of issues left open to discuss.

As future work, in order to gain a more in depth understanding of ways in which structured asynchronous discussions can support remote teams resolve open issues prior to negotiations, we are analyzing the broader context in which this causal relationship was observed. In particular, we are analyzing the negotiation meetings behavior, by measuring the conversational efficiency in terms of speaking turns and words, and the process, by classifying the types of turn (e.g., questions, agreements), exchanged to reach mutual agreement on issues. This will enable us to understand which factors in the computer-mediated collaborative process contributed to these results. We thus hope to draw more detailed guidelines on conducting structured asynchronous discussions in support of expensive but important synchronous requirements negotiations.

Acknowledgements

This work is partially supported by MiUR-Italy, under grant PRIN 2006 “METAMORPHOS”.

References

[1] R.L. Daft and R.H. Lengel, “Information Richness: A New Approach to Managerial Behaviour and Organizational Design”, in BM. Staw and L.L. Cummings (eds.), *Research in Organizational Behaviour*, CT JAI Press, Vol. 6, 1984, pp. 191-233.

[2] R.L. Daft and R.H. Lengel, “Organizational information requirements, media richness and structural design”, *Management Science*, Vol. 32, No. 5, 1986, pp. 554-571.

[3] D. Damian, F. Lanubile, and T. Mallardo, “The Role of Asynchronous Discussions in Increasing the Effectiveness of Remote Synchronous Requirements Negotiations”, *Proc. of the Int’l Conf. on Software Engineering (ICSE’06)*, ACM Press, 2006, pp. 917-920.

[4] D. Damian, F. Lanubile, and T. Mallardo, “An Empirical Study of the Impact of Asynchronous Discussions on Remote Synchronous Requirements Meetings”, *LNCS*, Vol. 3922, Springer-Verlag, 2006, pp. 155-169.

[5] D. Damian and D. Zowghi, “Requirements Engineering challenges in multi-site soft-ware development organizations”, *Requirements Engineering Journal*, Vol. 8, pp. 149-160, 2003.

[6] F. Henri, “Computer conferencing and content analysis”, *The Najaden papers*, Springer-Verlag, 1991, pp.117-136.

[7] IEEE Std IEEE-Std-830-1998, IEEE Recommended Practice for Software Requirements Specification, IEEE CS Press, 1998.

[8] F. Lanubile, T. Mallardo, and F. Calefato, “Tool Support for Geographically Dispersed Inspection Teams”, *Software Process: Improvement and Practice*, Vol. 8, No. 4, 2003, pp. 217-231.

[9] W.J. Lloyd., M.B. Rosson, and J.D. Arthur, “Effectiveness of Elicitation Techniques in Distributed Requirements Engineering”, *Proc. of the Int’l Conf. on Requirements Engineering (RE’02)*, 2002, pp. 311-318.

[10] L. Macaulay. *Requirements Engineering*. Springer, 1996.

[11] S.E. Poltrock and J. Grudin, “Videoconferencing: Recent Experiments and Reassessment”, *Proc. of the Int’l Conf. on System Sciences (HICSS-38)*, 2005.

[12] L.P. Robert and A.R. Dennis, “Paradox of Richness: A Cognitive Model of Media Choice”, *IEEE Transactions on Professional Communication*, Vol. 48, No. 1, 2005, pp.10-21.

[13] C. Wohlin et al., *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers, 2000.

The Effects of Distance, Experience, and Communication Structure on Requirements Awareness in Two Distributed Industrial Software Projects

Irwin Kwan, Daniela Damian, and Sabrina Marczak
Software Engineering Global Interaction lab (SEGAL)
University of Victoria
3800 Finnerty Road, Victoria, BC, Canada
{irwink, danielad, smarczak}@cs.uvic.ca

Abstract

In global software development, communication is difficult due to distance between sites. How effectively do team members distributed among multiple geographical locations become aware of changes and clarifications to requirements? In a case study of two different global software development projects, we observed how requirement analysts, developers, and testers maintain awareness of changes in the project. To gather data, we attended local and remote meetings, and conducted interviews of project team members. Based on our experience with these projects, we discuss the following awareness factors in software development: distance, experience of team members, and communication structure. We present the effects on awareness, and provide some lessons learned for global software development projects. We expect these lessons learned can be used by projects with similar settings.

1. Introduction

In software development, a software developer should keep aware of events that are occurring in a software development project. *Awareness*, in the context of software development, is whether a software developer working in a project has knowledge of events, such as changes to a requirement suggested by a customer, that occur in the project. The communication that informs a developer about an event is an awareness notification. This awareness becomes especially important in global software development, where distance can have an effect on the quality of communication among project team members [2, 10].

It is believed that informal communication is a strong contributor to awareness within a project [4]. Developers have been observed to spend a significant amount of time engaging in informal communication [11] and in group activities [9]. Lack of awareness is costly: A leader who does not maintain his knowledge of the application's design may be unable to manage the team effectively, and can lose contact with the customer [12]. A developer must be aware of the latest project developments and must remain synchronized with the information available to the rest of the team or problems with design, quality, and cost may occur.

This paper reports observations made during a case study regarding awareness in software development. We conducted a field study of two different globally-distributed industrial software projects in the Brazilian development centre of an American company. One project does distributed development divided among Brazil and United States (US) development offices, with its business clients in the US. The other project does co-located development in Brazil, with its business clients in the US.

The paper is organized as follows. We describe the context of the study, including the company and the projects, in Section 2. We describe the data collected in Section 3, and our observations in Section 4. We explain factors that may have affected awareness in Section 5 based on our observations and provide some lessons learned that may improve awareness in an organization in Section 6. We conclude the paper in Section 7.

2. Case Study Context

We conducted an observational study of two projects in the Brazilian subsidiary of an American company, which we will call ORG for confidentiality reasons. ORG has offices all over the world, including development centres in the US, Brazil, and India. ORG assembles and ships its products world-wide, and has an extensive I/T department to support its internal processes. The ORG unit in Brazil is an organization recognized as SW-CMM Level 2, and the unit is working on the definition of Level 3 processes. This is a global software process improvement initiative to align all the development processes. Each individual unit defines how to use the set of standard tools available in the organization. We observed two projects in the Brazil office, selected according to availability and easiness to reach distributed members. Both projects involved communication and collaboration with remote clients in the US. We describe the projects below; the project names have been changed to maintain anonymity.

Shipping System The Shipping System project (SHIP) updates and maintains an internal software product used by ORG to support its shipping process. The product is approximately seven years old, and is a critical component of the company's business. The teams are located in the US and in Brazil.

We were able to observe the last week of a maintenance release, and the first 6 weeks of an enhancement release. At the end of the observation period, the enhancement release had approximately 10 weeks in its schedule before deployment.

Support Applications The Support Applications project (APP) enhances and maintains a group of internal software products used in ORG by product management and sales. There are over 100 applications within this group, though about twenty are considered critical to the operation of the company. The applications are mature, and are undergoing regular enhancements and maintenance. The project contains a development group in Brazil, with the business partners in the United States.

We observed 6 weeks of a maintenance and enhancements release. At the end of the observation period, the release had 6 weeks remaining in its schedule before completion.

2.1. Project Team Organization

We examined the similarities and differences between two projects in an attempt to identify factors which may have had an effect on awareness in distributed development projects. We discuss the organization of the projects and provide a brief outline of their processes below.

Both projects build software that supports internal processes within the company. The requirements for the software come from internal clients, which we call *business partners* (BPs). BPs are employees of ORG, and may negotiate with external clients of ORG, may manage product portfolios, and may be users of the end product.

2.1.1 Shipping System: Experienced Team with a Decentralized Communication Structure

SHIP's teams are a development team distributed geographically across two sites, and a test team located in Brazil. The project manager, as well as the BPs, are in the US. There are five developers in the US, including the development lead with seven years experience, and the senior developer with five years of experience, allocated full-time to SHIP. There are two developers in Brazil allocated full-time to SHIP, including the Brazil development leader (Brazil dev lead) with four years experience, and two developers in Brazil allocated partially to SHIP. In addition, four contractors in a different building in Brazil are allocated in varying amounts to SHIP, though SHIP is their most important project. There are four testers fully allocated to SHIP, located in Brazil. There are two environment coordinators (ECs), one in Brazil, and one in US, who have to manage the development and test environments. The developers and testers coordinate closely with these ECs. The team members are working together for three years in average, and they had run many projects on Shipping System portfolio.

The BPs communicate primarily with the project manager, lead developer, and senior developer in US, as well as the Brazil dev lead. Developers and testers are not encouraged to contact the business partners directly unless it is through a team leader.

The team members in SHIP use face-to-face interaction, instant messenger, phone, and E-mail extensively; the latter three are used with remote team members. The Brazil dev lead often meets with the test lead and the EC face to face. The culture of the shipping system project encourages communication among team members, even across geographical boundaries (Figure 1). A team member is encouraged to contact any other team member for support. Because of this open communication among the team members, we consider SHIP as having a *decentralized communication structure*.

The SHIP team infrastructure features a version-control system for code, and a test management tool for defect-tracking, test cases and test results. Documents are stored on a shared folder and only leaders have write-access to it.

2.1.2 Support Applications: Inexperienced Team with a Centralized Communication Structure

APP is made up of 2 project managers (PMs), 5 business analysts, 7 testers, and 25 developers divided into 4 development teams in Brazil. One of these developers is the development lead coordinator (dev lead coordinator). In total, there are 39 people in the group. APP was formerly run by I/T employees in the US, but the company began an initiative to migrate application evolution to Brazil. Many of the applications were missing documentation, and many of the

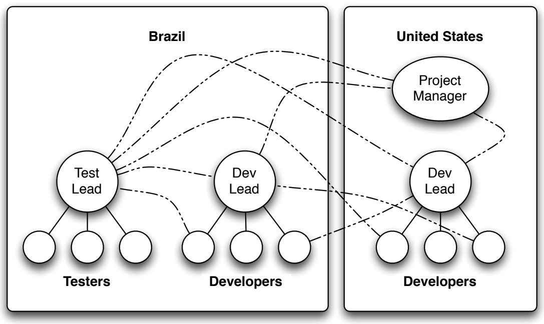


Figure 1. Organizational Structure of SHIP, with free communication among team members occurring between the Brazil and US sites.

employees who had originally developed the applications had been transferred to other groups, or had left the company. Consequently, every team member in APP is new to the product, with the most senior person having four months of experience in the project.

In this structure, no cross-team communication is supposed to occur without notification to the leaders of each team. The team leaders and managers enforce the policy that all cross-team E-mail communication must be CCed to the appropriate team leaders (Figure 2). The purpose of the CC is to ensure that the team leader can intervene in the discussion to provide advice or feedback. Because the organization is designed to control the flow of information through team leads, we consider APP as having a *centralized communication structure*.

As a part of its infrastructure, APP uses a central version-control system for requirements specifications and code. All documentation is checked into a repository.

3. Data Collection and Analysis

We observed the software developers involved in each project in local meetings and remote conference calls. We also conducted semi-structured and free interviews during a period of two months. The semi-structured interviews scripts were prepared after a short period of observations, and the free interviews were conducted according to events were taking place that called for the researchers' attention.

We conducted 14 interviews with the SHIP developers. We interviewed 2 US developers, the test lead's US mentor, and 5 different Brazilian team members. We held multiple interviews with the development leader (4) and the test leader (2) to keep updated with project events, and to receive clarifications on observations.

We conducted 13 recorded interviews with the APP contributors in Brazil. We contacted 4 developers, 2 project managers, 1 tester, and 5 business analysts. We conducted two interviews with the BA lead.

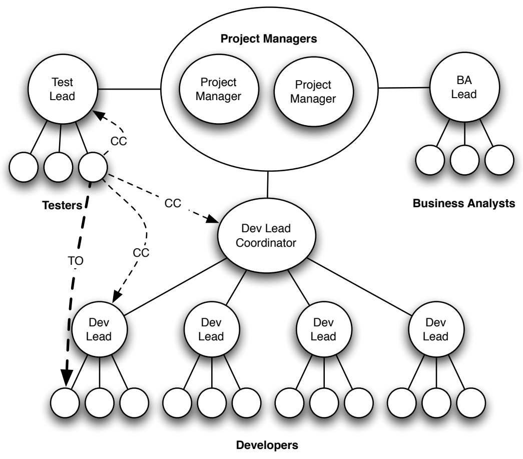


Figure 2. Organizational Structure of APP, with an example of a tester communicating with a developer

We were able to gather project-related E-mail messages from every member of the SHIP team members in Brazil except 3. We were able to gather E-mail from only five APP team members.

Through analysis of observations notes and interview transcriptions we identified the awareness issues presented as observations in the next section.

4. Observations of Awareness in Industrial Practice in Distributed Development

In this section, we discuss observed situations which highlight awareness issues we observed within SHIP. These observations do not stress all situations observed. No awareness issues within APP became apparent to us. This feeling was corroborated with team members during interviews. It does not exclude the chances the issues were hidden, but it means they did not impact the team. We discuss what the awareness issue was, how it was resolved, and the potential effects on the project had the issue not been resolved in a timely fashion.

Observation 1: Domain knowledge not shared

A contractor in SHIP did not receive a document containing domain knowledge related to one of his requirements, and consequently lost an afternoon of work. The requirement was to redesign a shipping label used by a client to meet localization requirements. Because he did not receive all of the information to code requirements, he designed prototype labels to send to the leaders for feedback. The Brazil dev lead had a document from the client that contained clarifications to a number of requirements, but he had forgotten to forward it to the developer when the contractor was assigned the new label requirement.

The document, which describes the client's updated label standard, had been sent to every development lead by a BP in 2005. The Brazil dev lead received a new E-mail message on the BP list, which mentioned the label standard, which prompted him to look up the document from past E-mail archives and discuss it with the US development lead and the US senior developer.

Just before a weekly team meeting, the Brazil dev lead was discussing the prototypes that had been provided with the US senior developer when he recalled the document describing the label standard. They both discovered that the contractor did not receive the updated label standard. The contractor found out about this document in the local team meeting between all team members in Brazil.

Observation 2: Requirements clarifications late

The test team of SHIP did not receive requirement clarifications from the US project manager on time, despite the fact that the development team received them. The test lead in Brazil recalled a situation when she sent a list of requirements questions from her team to the project manager. She had also CCed to a senior developer in the United States. When the project manager received clarifications, he informed the developers, but forgot to inform the testers.

The senior developer in the US who received the original copy of the message realized that the test team had not received these clarifications, and forwarded them in E-mail to the test leader, closing the awareness gap.

Although this situation was solved, a delay between sending the clarifications to the development team and sending the clarifications to the test team can mean that the test team works on outdated requirements which may lead to confusion especially when the development team and the test team synchronize.

Obervation 3: Deadline not communicated

A meeting discussing the deadline for the final day of the planning phase did not involve the test team. The test team of SHIP was not informed about the exit date for the planning phase that was being discussed among the project manager and some developers. However, an experienced test lead from the US who was not allocated to the project, but was acting as a mentor for the current test leader in Brazil, was present in a conference call when she noticed that the project manager was mentioning dates that seemed suspicious to her. She spoke with the current test lead about the situation, and then confronted the project manager to ensure that he knew he was forgetting the test team in his planning.

The awareness issue was resolved quickly by the mentor and prevented any damage to the project. However, had the test team not been made aware of the planning phase deadline, coordination in the project would have been affected. They may not have finished their estimations, their task assignments, or their requirements questions before development started.

5. Factors Affecting Awareness

We have identified some factors that may influence awareness in each project. These include (1) the effect of a distributed development, (2) the effect of experienced team members, and (3) the effect of communication structure.

5.1. Effect of Distributed Development on Awareness

Numerous sources have shown that distance between team members has a significant impact on communication [1, 8, 5]. SHIP was affected by awareness issues because of the distance between its two development teams. In Observation 2, coordination creates a large delay in communication: The test team leader must contact the project manager for requirements clarification over a large distance, and the project manager must respond. Already, the response is delayed, which causes problems with the test team's work. However, the issue is compounded because distance also reduces the project manager's awareness of the test team. In Observation 2 and 3, we see that the project manager does not immediately contact the test team, and therefore causes them to wait longer than would otherwise be necessary. Fortunately, the presence of experienced members on the SHIP team, as well as open communication among these members, helped to mitigate the damage that awareness problems may have caused.

APP did not have the same delays because the developers were all colocated. This may be a reason why we did not notice any awareness issues within APP. Only the clients were remote, but BAs contacted the customer in the US regularly and were able to promptly answer most of the developers' questions.

5.2. Effect of Team Member Experience on Awareness

It is well-known that experienced team members provide great benefits to software engineering projects. One benefit we have observed is their contribution to identifying and resolving awareness gaps.

In both systems, experienced members take the role of coordinators. In SHIP, the experienced team members worked alongside each other team member, but were kept up-to-date frequently using both formal and informal communication, even across geographically-distributed sites. We have observed that, in every awareness situation, the gap in awareness was bridged by an experienced member who realised that there was a knowledge gap, and took actions to resolve the gap. For instance, in Observation 1, the Brazilian lead, with the assistance of the US senior developer, identified that the contractor did not have the label standard. Observation 2, the senior developer from the US forwarded information to the test team when she received a reply from the project manager regarding the test team's questions. In Observation 3, the test leader's mentor, from the US, identified that there was a gap in awareness between the project manager, and the test team. We see in these cases that an experienced member had an intuitive sense of where awareness gaps existed, and took measures to fill those gaps.

In APP, experienced team members, serving as team leaders, were hubs of communication, and were made aware of every message sent across teams that were related to the requirements or the design. The team leaders were copied on each message so that they could intervene, thus avoiding awareness problems.

5.3. Effect of Communication Structure on Awareness

The way that team members coordinate, based on the communication structure, has an effect on how aware each team member is. The difference between the SHIP and the APP communication structures are vastly different, which may explain some of the awareness issues in SHIP.

The communication structure in SHIP is a decentralized structure. Although there are clearly-defined development leaders, the team members are not required to go through the development leaders when contacting each other, although there is extensive use of CCed E-mail. Because communication was not structured in this group, there were more communication paths (up to $\frac{(n-1)^2}{2}$ lines of communication [6]), and therefore, more possibilities for a gap in awareness. In Observation 1, the developer did not receive the label requirements document that was supposedly sent to each team member. Observation 2 and Observation 3 also highlight situations where a lack of structured communication caused the project manager to forget to contact the test leader. Despite the structure, the team was able to stay aware of its requirements, especially during the volatile planning phase, where requirements from the BPs came to the team very loosely-defined. We did not see these awareness problems in APP, where team leaders were able to intervene if they observed gaps in awareness.

In an interview with the development lead coordinator of APP, he stated that this structure is strong for new teams such as APP until they can gain more experience. He mentioned the volume of E-mail as a problem. The development lead coordinator also mentioned a vision for the team to move toward "more decentralized communication" as it gained more experience, and stated the advantages of decentralized communication as reduced communication on team leaders, and faster information exchange among team members. In this instance, we see a possible trade-off between how strict communication within a team is, and how aware team members in the team are.

6. Lessons Learned

Based on our experience as researchers with the case study, we have identified the following lessons in distributed software development. These lessons should be applicable to medium-sized distributed software projects with remote customers that have similar project settings.

6.1. Experienced Team Members Bridge Awareness Gaps

For most of the cases in which we observed an awareness gap, the gap was bridged by an experienced team member who had caught wind of the situation. In SHIP (the decentralized group), the senior developer from Brazil, and the senior developer from the US were able to intervene in order to provide information to team members.

Based on this observation, an organization should try to not only retain experienced team members in a team, but also make them as accessible as possible, especially if the experienced team member is remote.

6.2. Centralized Structures Keep New Teams Aware

We have seen that experienced team members contribute significantly in identifying and bridging awareness gaps. In APP, the team members were very new to the application domain, but yet, did not suffer from any awareness problems. In the team leaders were able to intervene if they observed gaps in awareness. Although we cannot claim that this was solely due to the use of a centralized structure, we believe that this may have had an influence in preventing awareness problems.

There are, however, some advantages to the decentralized structure. Although the decentralized structure experienced more awareness problems, we cannot claim that this structure is worse than a centralized structure, namely because the projects cannot be compared directly. We found that members in the decentralized team engaged in informal communication using E-mail and instant messenger frequently with members of the remote site—usually multiple times a day. This was in contrast to a study performed by Herbsleb and Grinter, which found that remote communication did not occur unless scheduled [8]. Other advantages to the decentralized structure include less dense communication and faster response times from colleagues, but the decentralized structure may risk information overload [3]. Although there were awareness problems in SHIP, the gaps were closed quickly, perhaps due to swift, unobstructed communication even to remote sites.

A project may wish to strike a balance between a decentralized and a centralized structure. However, in an organization with team members who are new to the domain, the team members should communicate every message to their appropriate team leader to ensure that leaders are aware of what solutions are being discussed. Team leaders can easily contribute to the discussion and remain aware of what the team members are working on.

6.3. Frequent Meetings Improve Awareness among Local and Remote Teams

Groups in ORG often use meetings to synchronize information among project members. Every person interviewed in the study mentioned the importance of regular meetings to keep aware of project events. The majority of the groups met twice a week to synchronize information among each other. In SHIP, the group had one weekly meeting for everyone at Brazil, and an additional meeting which included the developers from the US on a different day of the week. A group from APP met as often as five days a week, but scaled back to three days a week after the team leader received feedback that team members were not receiving enough new information during each meeting.

To maintain awareness and ensure that team members are up to date with the project's events, the local team members should meet face-to-face as a group, and remote team members should meet together despite the distance. ORG used conference calls for remote team meetings.

6.4. A Developer in a Distributed Team who has Multiple Tasks Reduces The Effect of Delays from Coordination

There is a significant delay when working in global software development [8, 7], but parallelizing work may help minimize the effect of this delay. When planning a project, especially in a global software project, a manager should consider assigning a number of stable requirements with an unstable requirement to the same developer. This will allow a developer to explore the unstable requirements and send appropriate questions to the remote team, or to the customer. While waiting, he can work on stable requirements and experience minimal downtime.

The developer in Observation 1 who did not receive the documentation from the lead developer told us that he was still productive even after he asked the business partners for requirements details because he had other tasks to do.

The effect of the increased delay in global software development may have been reduced because the developer could work on other tasks. Although some time was lost due to a lack of awareness about the requirement, the time may not have been significant because he was able to work on other, more stable requirements while waiting for more information.

7. Conclusion

Awareness ensures that each team member is up-to-date so he can do effective work. However, maintaining awareness in a distributed development environment is extremely difficult. From this case study of two distributed software projects, we observed the following factors as having an effect on awareness. (1) A project with distributed development sites reduces awareness. (2) Experienced team members bridge awareness gaps. (3) A centralized communication structure may prevent awareness problems, but at the cost of information overload [8].

There may be other factors that affect awareness in a software development environment that we have not observed. For example, requirements stability may be a factor. A project that stabilizes its requirements early reduces the need for communicating changes late in the project that may lead to awareness problems. Cultural issues may have an effect on communication and awareness. Our list of awareness factors is far from complete.

We present the following lessons in our study. (1) Experienced team members can bridge awareness gaps, so ensure they are accessible. (2) A centralized communication structure can help a new team keep aware. (3) Frequent meetings improve awareness and help detect awareness issues. (4) A developer in a distributed team with multiple tasks can be productive when there are delays.

This is by no means a comprehensive list of recommendations. This study is limited because the observations, though structured, are informal, and the sample size is small. These lessons may not be applicable to every distributed development project, but we believe that they are useful to those who wish to improve communication, coordination, and awareness in their projects.

References

[1] T. J. Allen. *Managing the Flow of Technology: Technology Transfer and the Dissemination of Technological Information Within the R&D Organization*. The MIT Press, January 1984.

[2] D. Damian. Stakeholders in global RE: Lessons learned from practice. *IEEE Software*, March 2007.

[3] D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the wild: Why communication breakdowns occur. In *Second International Conference on Global Software Engineering (ICGSE), Munich, Germany*, August 2007. To appear.

[4] D. E. Damian and D. Zowghi. The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization. In *IEEE Joint International Conference on Requirements Engineering 2002, September 9–13*, pages 319–328, September 2002.

[5] K. Ehrlich and K. Chang. Leveraging expertise in global software teams: Going outside boundaries. In *International Conference on Global Software Engineering 2006, Florianopolis, Brazil.*, pages 149–158, 2006.

[6] J. Frederick Phillips Brooks. *The Mythical Man-Month*. Addison-Wesley, 1975.

[7] J. D. Herbsleb and R. E. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*, 16(5):63–70, 1999.

[8] J. D. Herbsleb and R. E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *ICSE*, pages 85–95, 1999.

[9] J. D. Herbsleb, H. Klein, G. M. Olson, H. Brunner, J. S. Olson, and J. Harding. Object-oriented analysis and design in software project teams. *Human-Computer Interaction*, 10(2/3):249–293, 1995.

[10] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. An empirical study of global software development: Distance and speed. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 81–90. IEEE Computer Society, 2001.

[11] D. E. Perry, N. A. Staudenmayer, and L. G. Votta. People, organizations, and process improvement. *IEEE Software*, 11(4):36–45, 1994.

[12] Y. Yamauchi, J. Whalen, N. Ikeya, and E. Vinkhuyzen. The problem of knowledge decoupling in software development projects. In *28th International Conference on Software Engineering, Shanghai, China*, pages 877–880, 2006.

A Model of Requirements Engineering at Organizational Interfaces: An Empirical Study on Distributed Requirements Engineering.

Dorina-C. Gumm
University of Hamburg
Department of Informatics
Center for Architecture and Design of IT-Systems
gumm@informatik.uni-hamburg.de

Abstract

In this paper results are presented from an empirical study about the relationship between requirements engineering practice and distributed software project settings. The focus here lies on organizational distribution and the requirements engineering activities that take place between *organizational units. In order to understand distributed requirements engineering, the concept of organizational interfaces is introduced. Requirements engineering activities are then analyzed with respect to organizational interfaces. The resulting model aims at the facilitation of practitioners and researchers to design distributed processes and understand respective challenges.*

Keywords: Distributed requirements engineering, organizational distribution, organizational interfaces

1 Introduction

Distributed software development projects are characterized by the cooperation of more or less independent organizations or organizational units. The problems resulting from the communication and cooperation difficulties, from different development approaches used, and from the various cultures (national, social) and languages involved, are discussed in several recent papers e.g. [6, 13, 15, 2, 12].

In order to understand the interplay between project's distribution and requirements engineering in detail, an empirical study has been conducted. As basis for this study serves a taxonomy of distributed software development [11]. This taxonomy distinguishes between the phenomenon of distribution, respective challenges, and solutions to deal with the challenges, which all influence each other. The phenomenon of distribution is described by four dimensions: physical, temporal, organizational distribution and distribution among stakeholder groups. The taxonomy also implies the concept of *perceived distance* which illustrates that distribution not only depends on objective project settings but on the perception of individuals.

One goal of the empirical study is to relate requirements engineering challenges to these dimensions of distribution. Though this relation turned out to be difficult and less promising, the empirical data indicate that the organizational dimension of distribution is more crucial than the other dimensions. Similar results are presented by Berenbach [2]: In several large global projects "most of the issues could be traced back to problems with organizational structure and/or management".

Requirements engineering as a highly communicative task that needs to be conducted in cooperation with different stakeholders [14, 7, 8] actually takes place between organizational units. The empirical data provide rich material to understand this in detail. By using Grounded Theory for data analysis (see Section 2) a concept could be developed that helps understanding the nature of requirements engineering in distributed project settings. This concept is called *organizational interfaces* and describes what is *between* organizational units. Using this concept for analyzing requirements engineering practice, three types of activities could be identified: Activities for 1) requirements management at organizational interfaces, 2) increasing interface maturity and 3) designing organizational interfaces.

The paper is structured as follows: In the next Section I present research setting, method and empirical cases. The concept of organizational interfaces is presented in Section 3 and the types of activities in Section 4. The paper concludes with Section 5 by drawing a summary and pointing out to consequences for future research.

2 Empirical Study

The experience drawn on in this research comes from an interview study conducted with 9 industrial partners. The research goal of this study was to investigate the connection between distributed project settings and requirements engineering practice. The research work was carried out with participants of distributed software development projects and who are involved in requirements engineering in one or the other way.

In order to understand the correlation between distributed software development and requirements engineering, a series of semi-structured expert interviews [10] has been done. Rather than providing quantifiable responses to a specific question obtained from a large sampling of the population, expert interviews allow a deep insight in specific project settings and practice.

Grounded Theory [19] has been chosen as methodology to analyze and get access to the rich data collected during the interview study. Grounded Theory is rooted and accepted in social sciences and has several times been used for information systems research [16, 1].

The name "Grounded Theory" refers to theory that is derived from inductive analysis of and thus grounded in empirical data. Basic idea of this analysis approach is to read and re-read a textual database (such as field notes, transcribed interviews) and discover and label phenomena, categories, concepts, properties and their interrelationships [5].

The selection of industrial partners was driven by the goal of collecting experiences from a broad spectrum of distributed projects. The companies (introduced in Table 1) are located in different application domains and develop software for different target groups (like customer-specific software and kind of standard software).

2.1 The Case Projects

The participating industrial partners are large organizations from different sectors and are distributed in very different ways. The following paragraphs provide a brief description of the case companies, the respective software development project and its distribution characteristics according to [11].

Company A is an automotive manufacturer. The software under development is a control system for automobiles. In this specific case the company cooperates with two other automotive manufacturers that are located in the USA and Japan. Goal of this cooperation is to use the software for different products to increase the number of sold items. The project's distribution that is relevant for requirements engineering is caused by the fact that three persons are responsible for requirements specification, one at each site (Germany, USA, Japan). Requirements negotiation thus does not happen only between the development division and customer but in addition between the three sites and between the stakeholders at each site.

Company B is located in the sector of logistics and postal delivery. It develops a routing software which is an individual solution for one customer. The distribution here is caused by the customer who is dispersed over Germany and a third party software supplier located in Canada. Requirements need to be negotiated between users and analysts, between analysts and developers as well as between developers from this company and from the software supplier in Canada.

Company C develops software for digital rights management and operates in the telecommunication sector. The project's distribution here mainly refers to globally and organizationally distributed customers that have specific and often disjunctive or even contradicting requirements. In addition to the development and customers other stakeholders like consultants and marketing play a major role in the requirements engineering process.

Company D develops an airline business software. Similar to Case B a core distribution characteristic is that the software company uses third party software that is adapted and further developed for their own customers. A specific circumstance here is that the cooperation between developers and end-users had been poor for micro political reasons. Project-internal distribution relates to the fact that several key persons in the project are external staff.

Company E is also located in the automotive sector and in our study we focus on the development of the navigation part of an infotainment system. The company consists of many sub-companies that represent specific knowledge centers and that are dispersed all over Germany. These sites operate rather independently but need to cooperate for specific software development projects which causes, in addition to physical distribution, also an organizational one.

Case F is about a hospital; its IT department uses a third party software to adopt, customize and enhance it for this hospital. The distribution perceived as challenge here refers on one hand to the cooperation between IT department and software supplier and on the other hand to the organizational distribution of end users. The end users are located in different hospital wards and have thus different and sometimes contradicting requirements.

Company G is a globally operating consulting firm which conducts global software development projects. In the respective interview a variety of projects have been covered to discuss distributed requirements engineering practice. However, all discussed projects developed customer-specific business software. The distribution refers mainly to distributed development teams and to customers that are typically located far from the development team.

Company H is also located in the telecommunication sector; this case is about the development of a demo version of their content management system. This demo version is used by their consultants to present their CMS to potential customers. In this project the development group was dis-

Cases	Sector	Product
A	Automotive	Control System
B	Logistics / Post Delivery	Routing
C	Telco	DRM
D	Airline	Airline Business System
E	Automotive	Navigation
F	Health Care	Hospital Information System
G	several	several information systems
H	Telco	CMS / Demo Version
I	Research	particle accelerator

Table 1. Case Companies

persed over two locations in Germany. Even more important was the organizational distribution of groups that provide requirements for the software. Requirements sources in this case have been projects that developed former demo products as well as projects working on specific aspects of the actual CMS; in addition the dispersed group of consultants was involved in that project. Also the temporal distribution has been a key issue since most of the project members were not involved full time.

Case I is about a research center for particle physics. The respective project was about requirements definition for a particle accelerator. Even if it is not a software development project, software requirements engineering methodology has been used to carry it out. The project was affected on one hand by the organizational distribution of the requirements stating units and on the other hand by the temporal distribution caused by stakeholders who are not available during the whole project life cycle. The benefits and challenges of the project are very similar to the other cases.

To classify types of distributed projects, Paasivaara [17] proposes the distinction between inter-organizational and intra-organizational distribution. This distinction is useful; however, in the empirical data a third type of distribution turned out to be relevant for requirements engineering: inter-project distribution. In the following I briefly present some typical structures of the case projects according to the mentioned types of organizational distribution:

- *Inter-organizational distribution.* In some of the cases inter-organizational distribution is given because the studied development project cooperates with a third-party software vendor (B, D, F). In these cases, the third-party software needs to be further developed, integrated into other software, and/or customized. The cooperation of more or less equal partners to increase competitiveness can also lead to inter-organizational distribution (A). Inter-organizational distribution is also given if the project deals with customers from various companies or even application domains (C, D, E).
- *Intra-organizational distribution.* Intra-organizational distribution can be observed in cases where a large system is developed by a variety of rather independent technical departments (A, E); or where requirements holders are dispersed among several departments (I). In one case, different user groups with divergent requirements are located in different organizational units (F).
- *Inter-project distribution (both inter- and intra-organizational).* In a variety of cases the distribution among projects plays a major role for requirements engineering. Inter-project distribution is given when the studied software development takes place across several related projects. This could be observed in cases where other projects develop software that needs to exchange data with the actual software (A, B, C, E, H). Inter-project distribution is also given in cases where parallel to the actual project other projects start or end in which former or later versions or branches of the software is developed (A, B, H).

The selected case projects illustrate the multifariousness of organizational distribution. Therefore, the empirical data have been analyzed with respect to underlying similarities, for which Grounded Theory proved to be very helpful. As result I present the concept of organizational interfaces in the following Section 3; this concept is used afterwards for describing distributed requirements engineering practice in Section 4.

3 Organizational Interfaces

Requirements engineering is a highly communicative task in the software development process for which a variety of stakeholder groups and organizational units need to cooperate [14]. Thus, many RE activities take place *between* organizational units; this *between* can be described with the concept of *organizational interfaces*.

3.1 Definition

An interface is a part of a system at which independent systems meet and act on or communicate with each other. For example, software interfaces are logic points of contact and define the exchange of commands or data. Thus, an interface can be defined by its connected endings (systems) and the data exchange, respectively.

An organizational interface is defined here by two organizational units which interact within a project. Organizational unit is a place holder for any organizational structure a group of stakeholders is working in. Organizational units might be units within an organization, like IT or marketing departments; they could also represent cooperating companies like a software vendor and a company further developing this software. Organizational units also can represent different development projects, hierarchy levels or functional roles.

Typical organizational interfaces in software development projects are, for example, between the development and the customers, between development and the users or between development and analysts which are in contact with the users. Most of the case projects are far more complicated and thus they deal with a lot of organizational interfaces (also compare [3]). In the empirical data most frequently the following interfaces play an important role for requirements engineering: the interfaces between a) the actual development and a third-party software vendor; b) cooperating technical divisions; c) the actual development and one or more customers.

Interfaces always exist between organizational units and are no specialty of distributed projects. However, in physically distributed software development projects, organizational distribution becomes a bigger challenge. This is due to the fact that in physically distributed projects organizational interfaces a) become more visible than in non-distributed projects, b) are more perceived as interfaces and c) they hence more likely lead to problems.

3.2 Characteristics

According to the empirical cases, the perceived distance between organizational units highly depends on the established communication and cooperation at the respective interface and thus on the *interface maturity*. Communication and cooperation at different organizational interfaces differ significantly in the cases. These differences can be identified between different interfaces in one case project as well as between equivalent interfaces in different case projects.

Thus, the empirical data have been analyzed concerning the similarities between the different interfaces to develop a general notion of organizational interfaces that is independent of particular projects or interfaces. According to this analysis, interface maturity can be described by means of the following two interface characteristics: communication channels and social distance.

Communication channels. Communication channels represent the media that are used for communication at an interface. In the empirical data three types of communication channels could be identified:

- *Contact person*: In many cases communication is channelized by one contact person per organizational unit. Such persons are, for example, responsible for collecting requirements at several interfaces in order to transfer them afterwards to other members of her/his organizational unit, or for transferring information in the other direction.

- *Social events*: Often communication at interfaces is arranged by events like workshops, stand-up-meetings or user events (at professional exhibitions or prototype presentations).

- *Artifacts*: Artifacts like requirements documents or requirement management software are often used for channelizing requirements related issues.

Some examples illustrate the importance of communication channels. In Case B the communication at the interface developers–users was mediated by analysts. In Case F key users served as communication channel between IT department and single hospital wards.

Regular meetings and workshops are the most frequently used channels for communication at various interfaces. Especially in cases with a large and, if so, anonymous user group special social events are conducted. The two interviewees from the automotive sector discussed so-called car clinic events where key users are invited to evaluate newest prototypes. In two cases professional fairs are an important communication channel between development and anonymous users. In another case special conferences held by the software vendor served as communication channel.

Communication at a specific interface is (if at all) usually supported by a collection of communication channels. However, not only the adequate channel is crucial for cooperation but also the degree by which this channel is established and used. Thus, the interface maturity regarding communication channels is dependent on two dimensions:

- Amount (many channels – no channel)

- Development (well established – poorly established)

Social Distance. Social distance represents the understanding that exists between two organizational units (freely adapted from [4]). In the empirical data social distance can result from differences regarding

- *culture* (both national and working culture): Different cultures rooting in nationalities or working habits is often accompanied by a perception of distance; at least until the involved parties carefully deal with it and mutually learn from each other.
- *language* (both national and professional language): Also different languages affect social distance. In the case projects requirements engineering suffers from differing national languages as well as from different vocabulary in the same language of cooperating units.
- *views on a subject*: In a variety of cases the involved organizational units hold different views on the subject matter which causes irritations and thus a perception of distance.
- *processes*: Differing processes at an interface are a very crucial issue that can lead to social distance because they can imply different notions of requirements engineering as well as on scheduling.

Interface maturity is affected by social distance but not always in the same way. According to the empirical data interface maturity does not necessarily increase by decreasing social distance or the respective diversity. Rather, interface maturity depends on how sensitive the organizational units deal with it.

The concept of organizational interfaces does not describe organizational structure in general but focusses on the *between* of cooperating organizational units. This focus is of particular interest for requirements engineering since the latter is one of the most communication-intensive tasks in software development. The concept is illustrated in Figure 1.

The concept of organizational interfaces helps to understand the relation between requirements engineering practice and distributed project settings. Requirements engineering activities identified in the empirical data can be analyzed using the concept of organizational interfaces. The results are presented in the next section.

4 RE Activities at Organizational Interfaces

Requirements engineering at organizational interfaces not only implies activities like requirements elicitation, analysis, negotiation or change management. Rather, those activities are intertwined with activities referring to organizational structures and thus to interfaces. The interviewees always foucssed on the RE activities between the units, not those within.

Thus, for understanding requirements engineering in distributed projects it is useful to be aware of activities that are closely connected to the respective interface. According to the empirical data the following categories of activities could be identified, which are described in the following paragraphs:

1. Requirements Management;
2. Increasing Interfaces Maturity;
3. Designing Organizational Interfaces;

4.1 Requirements Management

Four types of requirements management activities that refer to interfaces could be identified: requirements bundling, requirements redistribution, requirements adjustment, and informing.

Requirements bundling. In many cases requirements elicitation is (supposed to be) done within the requirements holders' organizational unit. If requirements holders are dispersed over several organizational units, the task of bundling requirements becomes very important. Requirements are collected at several interfaces and bundled at a central location. This task works best in the cases where one person of the requirements bundling organizational unit is responsible for it. If so, this person serves as communication channel at the respective interfaces.

Two examples illustrate this: In Case F each hospital ward provides one key user who bundles requirements stated by end users; and one consultant of the IT department collects the requirements from various hospital wards.

In Case E the person responsible for specification must carry together requirements stated by professional departments. The interviewee came to the point:

> Elicitation only works by [...] collecting them from different stakeholders. [...] And I think it is necessary to collect them distributed. The problem I definitely see in bundling them.

Requirements redistribution. Redistribution of requirements (and respective changes) is subsequent to requirements bundling. Especially when developing large systems, requirements have to be analyzed according to which parts of the system (and respective development groups) are affected by them. System interface requirements typically affect more than one development group. The task of redistributing requirements at respective interfaces needs to

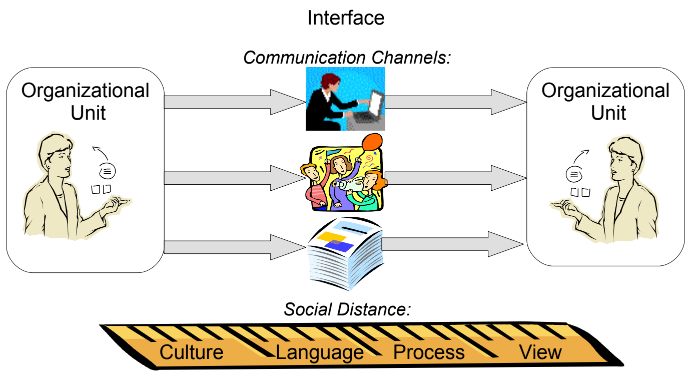


Figure 1. Characteristics of Organizational Interfaces

be carefully performed but often suffers from insufficient qualification of the responsible person.

In Case E it is an important task of the project leader to redistribute parts of the requirements set to responsible development divisions. Often requirements affect more than one development division. Also in Case C the task of requirements redistribution has been discussed; in contrast to case E requirements were redistributed to cooperating projects, not company divisions.

Requirements adjustment. In many cases the respective software needs to be embedded in an infrastructure of existing and/or other new software. Hence requirements for the actual software needs to be adjusted to requirements of software developed or maintained in other projects. The need for adjustment can be observed in inter-organizational as well as intra-organizational distributed development efforts.

In Case E this adjustment takes place, for example, between the three cooperating automobile manufacturers. Here, three different views on the product need to be synchronized.

In Case B the software under development needs to blend well with other systems for data exchange. This is similar to Case C where requirements of the own digital rights management software and those of vendors' or customers' software (e.g. billing systems) need to be adjusted.

Especially in cases with more than one customer a very important activity is to balance requirements that are conflicting or go beyond the scope of the product. This turned out to be important especially in Cases C and E. On one hand requirements should be customer-independent to satisfy as many customers as possible; and a close cooperation with one customer could lead to a too specific product. On the other hand, a close cooperation makes it possible to discover the 'real' needs. As one interviewee illustrates:

> We try to be as independent from a customer as possible. And we also profit from [the fact] that we work very closely together with the customers, because [like this] we learn very very much from it about what are real requirements.

Requirements adjustment is also accompanied by the negotiation of concerns between development and customers. If during development questions arise about specific requirements these need to be answered by responsible persons. In Case G, for example, this is processed by using an issue list in which developers can enter their concerns. If the functional designer is not able to clarify this point (for example because there is a contradiction to other requirements), he or she needs to go back to the customer to negotiate this.

Informing. A variety of requirements engineering activities simply refer to information transfer from one organizational unit to another. For example, at the interface between development and customers informing mainly refers to document exchange, especially requirements and technical specifications, for keeping each other up-to-date. At the interface between different development departments much effort is necessary to inform each other about changes that may affect requirements. Informing also implies preparing information for different stakeholder groups with differing knowledge, background and expectations.

In Case A new versions of the requirements specification are regularly exchanged between development and customer. Basically, the contractor regularly receives new versions.

In Case C a so-called Jour Fixe had been established where not only customers and developers take part but also sales persons or consultants to inform each others about requirements, change requests and requirements implementation status. In Case H it was important to regularly inform customers about what will (not) be included in the next software release.

Requirements management activities at organizational interfaces are illustrated in Figure 2.

4.2 Increasing Interface Maturity

Whereas the activities discussed in the last subsection mainly refer to the handling of requirements at organizational interfaces, a variety of other activities refer to increasing the interface maturity in order to improve conducting the other activities. Such activities regard the strengthening of communication channels as well as the reduction of social distance.

One could argument that such activities belong to project management rather than to requirements engineering [9]. However, in all case projects, requirements engineering activities and interface-related activities are very tightly intertwined. A separation of these types of activities makes it more difficult to understand requirements engineering in distributed project settings.

Strengthening Communication channels In a variety of case projects requirements engineering suffers from poorly established or poorly used communication channels. Thus, activities could be observed that refer to this leakage which is often accompanied by poorly defined responsibilities. Poorly defined responsibilities cause problems such as that requirements cannot be bundled and that hence they get lost; that information about requirements changes are difficult to trace and transfer; or that requirements specification is delegated to unqualified personnel.

Strengthening communication channels can also mean to improve requirements management software usage or to communicate new ways of using it.

In Case I requirements had to be elicited from a variety of different technical divisions. Since the group that was responsible for the general RE process was not able to define requirements of these different technical areas, they tried to define persons responsible for requirements definition in each division. However, for some of them requirements definition was too unimportant so that they delegated the task to someone less qualified. Requirements engineering became a task of looking for qualified persons and, if necessary, of mediating between less qualified individuals and actually responsible persons. However, all in all the strategy of defining responsible persons for specification was perceived as good strategy to increase respective interface maturity.

Also the interviewee of Case D complained about poor defined responsibilities. Requirements where stated by everyone who thought to contribute; this resulted in the fact that some important persons did not contribute at all, that other persons contributed who did not need to, and that requirements occurred uncontrolled.

Decreasing Social Distance. Interface maturity also depends on how good the respective organizational units understand each others' views and approaches. Thus, requirements engineering implies activities of mutual learning about requirements, needs, views and technical options as well as about differing processes. Activities in the case projects imply conducting workshops to get to know each other; developing a shared vocabulary; deepening the communication to learn from each other; and agreeing on differing processes. In most of the cases the latter issue does not necessarily mean to share a common process but to synchronize processes and to be aware of the differences (cf. similar results in [18]).

For example in Case B, specifications were poor due to misunderstandings. Development and customer's analysts hold different notions of which features the system should provide. Thus requirements engineering comprised activities to facilitate getting to know each other, mutual understanding and developing a shared vocabulary.

In Case C requirements engineering with one particular customer turned out to be as important as critical. The customer is of high value, inter alia because they analyze and discuss requirements in a very detailed manner. Since this customer is located in Japan whereas the development is located in Germany, difficulties occurred regarding cultural differences and hence different notions of the requirements engineering process and of the role of a 'software developer'. Some time and effort was necessary to learn about each other's conception of how the requirements discussion should proceed.

4.3 Designing Organizational Interfaces

In many cases organizational interfaces need to be established in the first place. The most striking example can be observed in Case D where for micro-political reasons no organizational interface between development and users exist at all. This shortfall hampered requirements elicitation and modeling and eventually led to wrong requirements. Only after the appearance of such problems, a respective interface had been established and users had been involved.

Not only the development of new interfaces is an issue – also the deletion or by-passing of existing interfaces turned out to be a frequent activity. This was often necessary to speed up the process or to facilitate mutual learning. A very illustrative example can be found in Case F. The hospital's IT department cooperated with the involved software vendor via their professional service. In order to increase the priority of the hospital's requirements, the IT department wrangled to establish a close cooperation with the software

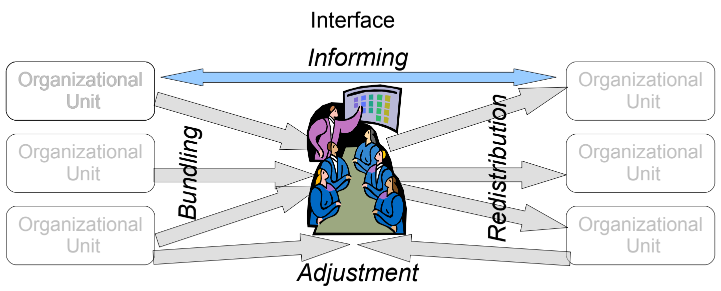


Figure 2. Characteristics of Organizational Interfaces

vendor's developers. This close cooperation also implied that the hospital became beta tester and reference hospital for the respective software.

In some cases the design of organizational interfaces implies the identification of responsible persons for being contact person for requirements issues. In Case E the (sub-)project managers are dedicated to be responsible for requirements engineering; and everything regarding requirements has to be carried out via these persons.

Interesting for the task of designing organizational interfaces is the Janus face of it. Whereas from some perspectives a new organizational interface is desirable for good requirements engineering, they are perceived as drawback from other perspectives because efforts of defining specific processes and communication channels might be foiled.

5 Conclusion and Future Work

In this paper I presented an empirical study of distributed requirements engineering. The problems discussed in the interviews mainly refer to organizational distribution. In order to understand the nature of organizational distribution and requirements engineering practice respectively, I first introduced the concept of organizational interfaces and then described how requirements engineering activities are related to organizational interfaces.

This work is supposed to support practitioners as well as researchers who have to deal with requirements engineering in distributed software development projects. The concept of organizational interfaces can help to understand what is necessary for conducting requirements engineering between organizational units, and to analyze which interfaces are important or need special attention in a particular project.

The different types of activities can help to understand how requirements engineering is embedded in and affected by distributed project structures. In addition it illustrates that much requirements engineering effort is spent on designing interfaces and increasing their maturity. Some empirical data indicate that such effort would not be necessary if the related companies have a larger expertise with (global) distributed projects. However, other data show that software development projects usually have not much time to carefully establish organizational interfaces, mutual understanding and common processes before requirements engineering starts. Thus, the results of this paper may help to understand distributed requirements engineering and to use such insights to carefully design the own project's approach.

The empirical data the results base on are limited as they are taken from only nine interviews. Though the small number of interviews and projects is not a representative data base to be unrestrictedly generalized, the Grounded Theory-driven analysis of the empirical data provide us with a deep understanding of the analyzed cases and direct our focus to concepts that abstract from particular cases. Also for this reason, the presented activities are not related to specific case projects or organizational interfaces.

Further work should be done for improving and validating the presented results. Currently they are examined by means of a longitudinal case study.

References

[1] D. E. Avison, F. Lau, M. D. Myers, and P. A. Nielsen. Action research. *Commun. ACM*, 42(1):94–97, 1999.

[2] B. Berenbach. Impact of organizational structure on distributed requirements engineering processes: lessons learned. In *GSD '06: Proceedings of the 2006 international workshop on Global software development for the practitioner*, pages 15–19, New York, NY, USA, 2006. ACM Press.

[3] J. M. Bhat, M. Gupta, and S. N. Murthy. Overcoming requirements engineering challenges: Lessons learned from offshore outsourcing. *IEEE Software. Special Issue on GSD*, 23(5):38–44, 2006.

[4] E. Bogardus. A social distance scale. *Sociology and Social Research*, 17:265–271, 1933.

[5] S. Borgatti. Introduction to grounded theory. 2003. Last access February 2007.

[6] K. Coar. The sun never sets on distributed development. *Distributed Development*, 1(9):32–39, January 2004.

[7] J. Coughlan and R. D. Macredie. Effective communication in requirements elicitation: A comparison of methodologies. *Requirements Engineering*, 7(2):47–60, 2002.

[8] D. E. Damian, A. Eberlein, M. L. Shaw, and B. R. Gaines. An exploratory study of facilitation in distributed requirements engineering. *Requirements Engineering*, 8(1):23–41, Feb 2003.

[9] R. Fahney, A. Herrmann, and R. Weibach. A new dimension in the distinction between Requirements Engineering and Project Management. Report from the RE&PM Working Group (belonging to the German Informatics Society's (GI) "Requirements Engineering" special interest group, 2006.

[10] U. Flick. *Qualitative Forschung: Theorien, Methoden, Anwendung in Psychologie und Sozialwissenschaften*. Rowohlt-Taschenbuch-Verlag, Hamburg, 4 edition, 1999.

[11] D.-C. Gumm. Distributed software development – a taxonomy. *IEEE Software. Special Issue on GSD*, 23(5):45–51, 2006.

[12] J. Hanisch and B. Corbitt. Requirements engineering during global software development: Some impediments to the requirements engineering process - a case study. In n/a, editor, *Proceedings of the European Conference on Information Systems ECIS'04*, page n/a, 2004.

[13] S. Krishna, S. Sahay, and G. Walsham. Managing cross-cultural issues in global software outsourcing. *Communications of the ACM*, 47(4):62–66, Apr. 2004.

[14] L. A. Macaulay. *Requirements Engineering*. Applied Computing. Springer Verlag London, 1996.

[15] J. S. Olson and G. M. Olson. Culture surprises in remote software development teams. *Queue*, 1(9):52–59, 2004.

[16] W. J. Orlikowski. CASE tools as organizational change: Investigating incremental and radical changes in systems developement. *MIS Quarterly*, 17(3):309–340, 1993.

[17] M. Paasivaara. Communication needs, practices and supporting structures in global inter-organizational software development projects. In *Proceedings of the ICSE Internaltional Workshop on Global Software Development*, 2003.

[18] M. Paasivaara and C. Lassenius. Collaboration practices in global inter-organizational software development projects. *Software Process Improvement and Practice*, 8:183–199, 2003.

[19] A. Strauss and J. Corbin. *Grounded Theory: Grundlagen Qualitativer Sozialforschung (engl.: Basics of Qualitative Research: Grounded Theory Procedures and Techniques)*. BELTZ, 1996.

Release Planning in Distributed Projects

Korbinian Herrmann
Technische Universität München
Fakultät für Informatik
Lehrstuhl für Angewandte Softwaretechnik
herrmann@in.tum.de

Abstract

During release planning project managers decide what to deliver in a specific release. In globally distributed projects his decision is a "wicked problem" [5] because local decisions might contradict global decisions. Project managers have to respect many factors such as customer preferences, time and budget as well as constraints from development. Existing approaches to release planning neglect the influence of system models [12].

This paper proposes a single tool that supports both release planning and modeling in globally distributed software projects. The approach allows developing a single model for every release. This enables project managers to make an informed decision with respect to the system model during release planning.

1. Introduction

A release is a version of a software system that will be made available externally [2]. During release planning project managers decide what to deliver in a release, e.g. they define the requirements of a release. Typically, it is not possible to bring all possibly feasible requirements to market within the next release. Requirements that shall be delivered within a given budget must be selected. This selection is relevant for the profit of the company: Bringing a new innovative feature to the market in time and budget might increase the profit enormously. [5]

Many sources influence release planning. Figure 1 gives an overview: Marketing, Support and Analysis result in new or changed requirements that could be implemented in the next release. Testing results in a set of bugs, which should be fixed. The project manager has to identify an optimal set of requirements to implement and bugs to fix within available resources: Marketing specifies a schedule. Project organization manages available project participants and their abilities. Strategical business planning dictates the budget. Risk management identifies risks related to the requirements that should be avoided.

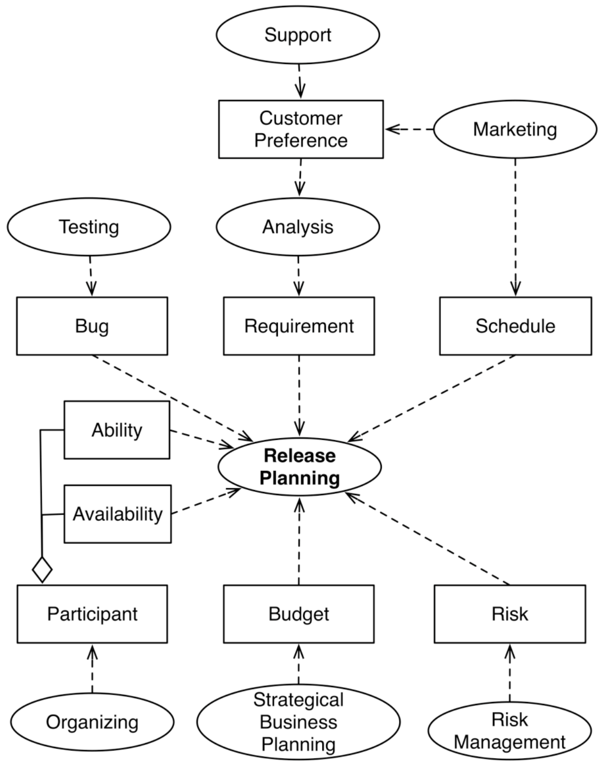


Figure 1: Sources of release planning

In global software engineering release planning is a "wicked problem" [5] because project managers can't cope with the complexity of considering all these sources from different sites: Requirements and bugs are identified, globally distributed, and limitations of resources are influenced from different locations. Furthermore, requirements are often still not clearly specified at project startup. As a consequence, release plans start evolving during requirements elicitation. Here, modeling influences release plans. This paper addresses this problem. It proposes a single tool for modeling and release planning. It provides consistency between release plan and model, which increases awareness of modeling and planning activities in different sites.

This paper is organized as follows: Section 2 describes existing approaches to release planning. Section 3 describes the approach of this paper. Section 4 introduces SYSIPHUS, which is the basis for the proposed tool. Section 5 explains how to model multiple releases. Section 6 focuses on the selection of requirements for a release. Section 7 shows how project managers and developers collaborate being aware of each other. Section 8 shows how traceability supports release planning and modeling. Finally, section 9 concludes the paper with a perspective on future work.

2. Related Work

Existing methods for release planning such as "EVOLVE" [7] try to generate an optimal release plan considering criteria and dependencies of requirements. Each method has its focus on specific criteria, for instance the method "Planning Software Evolution with Risk Management" [6] concentrates on minimization of the risks. None of these methods can produce the optimal release plan because the problem of selecting the optimal set of requirements is NP-hard [5][1]. Finally, it is the project manager who has to cope with selecting requirements for a release [5]. Existing tools such as the ReleasePlanner [10] [11], VersionOne [14] and the MicroTool [9] try to support project managers.

The ReleasePlanner [11] allows project manager to define criteria. Multiple stakeholders assess requirements considering these criteria. The ReleasePlanner uses this input to generate a set of suboptimal release plans. The project manager finally has to evaluate these alternatives. VersionOne [14] is an agile project management tool that supports planning of requirements. MicroTool [9] allows modeling requirements and assigning them to releases. It does not support the project manager in making his decision well informed.

Existing tools support either modeling of requirements or making an informed decision. As a result the current practice is to use a set of tools in a single software project, each of them supporting a single task. These tools are independent of each other and have their own repository to store project data. As a consequence these repositories are inconsistent and not up-to-date [8][15]. Consistency between system models and the release plan is important as models change quickly: For example if analysts get a better understanding of the problem domain, they add or change requirements in the system model. [3] Consequently, a project manager might have to review the release plan to prove that the changed or extended set of requirements can be done with available resources. Inconsistent models and release plans may lead to wrong decisions of project managers as well as to wrong models of developers.

3. Approach

This paper proposes a single tool for modeling and planning multiple releases. It deals with change providing awareness and traceability.

Existing tools only permit to develop one model at a time. The tool presented in this paper provides a single system model, called release model, for each release: It consists of a functional, a static and a dynamic model. Consider a simple portable Multimedia Player as an example. It is delivered in two consecutive releases: Release 1 and 2. In release 1 the Multimedia Player is only able to play songs; the user can transfer them using a USB cable. In release 2 its functionality is extended to support podcasts and bluetooth. The functional model of release 1 will only contain the use cases PlaySong and TransferViaUSB. The functional model of release 2 however, will contain the use cases of release 1 and 2: PlaySong, PlayPodcast, TransferViaUSB and TransferViaBluetooth.

The proposed tool uses release plans to define a release. A release plan contains a number of selected release items. Release items are entities that can be developed or solved in a release, e.g. functional and nonfunctional requirements, bugs or open issues. Alternative release plans support project managers in distributed projects: Every development site could elaborate one proposal for a release plan. Finally the project manager has to find a global decision respecting alternatives from different sites. Here, the tool supports his selection by providing criteria: These criteria may include the needed resources to develop the release, business criteria (e.g. profit), organizational criteria (e.g. skills of project participants) as well as criteria derived from the system model and architecture (e.g. number of off-the-shelf components).

To bridge the gap between developing and planning, the tool provides awareness of changes: Therefore it holds the model and releases consistent, i.e. release items of the release plan are in the release model and vice versa. Thereby, project managers respect the latest results of analysis and testing, while developers are aware of changes in the release plan.

Traceability supports project managers and developers to deal with change: For example, if a new bug is identified, project managers can find out the requirements, which this bug affects.

4. SYSIPHUS

SYSIPHUS [11] is a distributed application that emphasizes system models, collaboration models and organizational models equally. System models depict aspects of the system; examples are problem statements, requirements and detailed class models. Collaboration models include informal comments, issues, risks, and action items. Organizational models describe the participants and their relationship among each other. [3]

SYSIPHUS represents system, collaboration and organizational models of a project in a single graph. This unified representation allows offering “the same set of traceability” [3] (see Section 4) “and awareness” [3] (see Section 7) “services for all three types of models” [3]. SYSIPHUS provides a server to store the graph in a single shared repository. Participants access the repository through a variety of tools based on their skills and their role. An example for such a tool is the fat client RAT (see Figure 2) [15].

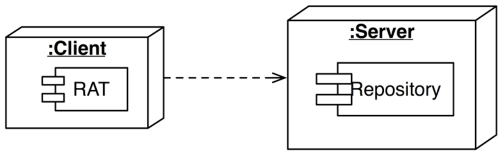


Figure 2: Components of SYSIPHUS [15]

A uniform meta model builds the SYSIPHUS graph to represent the models. This meta model consists of two classes: Model element and model link. These classes provide generic and extensible mechanisms to store them persistently, to control access to them and to track their history. „All system model elements (e.g. use cases, nonfunctional requirements), collaboration artifacts (e.g. comments, issues, action items), and organizational models (e.g., participants, teams) extend model element. Associations between elements are implemented by extending model link“ [3], which in turn extends model element (see Figure 3). The next sections describe selected system models, collaboration models and organizational models as well as their associations [3].

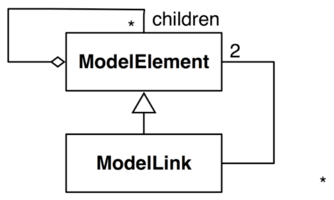


Figure 3: Meta model of SYSIPHUS [3]

4.1. System Models

SYSIPHUS provides system models for a number of development activities. To support analysis, for instance, these models include functional and nonfunctional requirements, features, actors, scenarios, use cases, as well as use case, class, sequence, activity and state chart diagrams. SYSIPHUS uses UML for system models and extends it with additional model element and model link classes. Figure 4 depicts some system model elements as an example: A use case is associated with classes that implement the use case. Diagrams represent model elements graphically. A diagram has an association to the model element

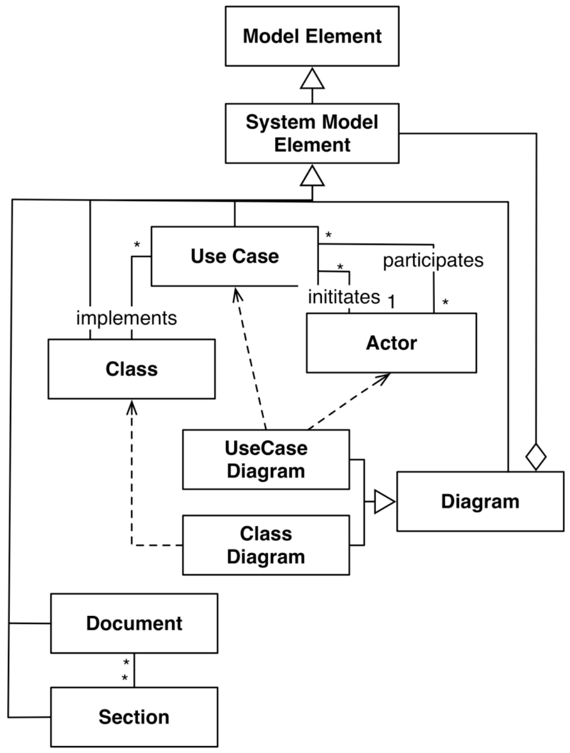


Figure 4: Selected system models [3]

they contain: Class diagrams consist of a number of classes while use case diagrams consist of actors and use cases. [3]

Documents are system model elements that provide a mechanism to show a structured view of the graph. A document consists of sections and subsections. Subsections contain a filter to insert model elements into a document. This allows mapping system, collaboration and organizational models to documents in the same way. [3]

4.2. Collaboration Model

SYSIPHUS supports collaboration by annotating model elements. Annotations are comments, action items, issues, and risks. By annotating collaboration applies to one or more model elements. Comments are a simple, unstructured way for participants to communicate. Action items represent a simple task model, which allows assigning tasks to project participants. Issues are part of the rationale management [4] and allow a structured discussion of open questions or problems by the participants using proposals and criteria. The participants assess the proposals of an issue using criteria. Risks are a specialization of issues and might threaten a model element and have a probability and an impact (see Figure 5). [3]

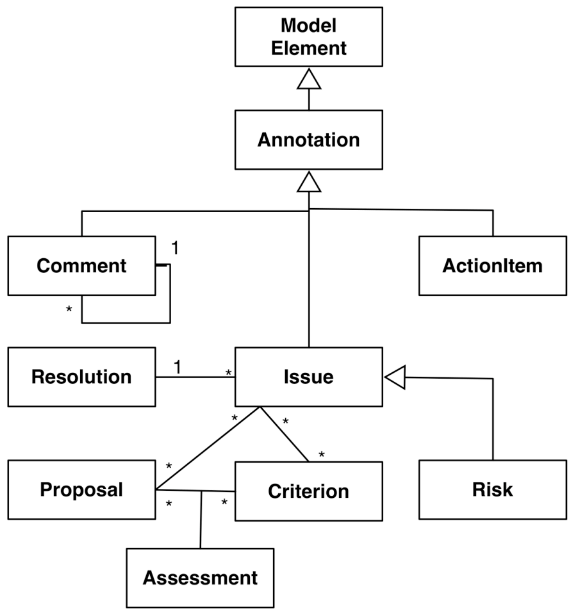


Figure 5: Selected collaboration models [3]

4.3. Organizational Model

The organizational model of SYSIPHUS consists of organizational units, which can be either participants or teams. A team has many organizational units (i.e. other teams or participants) as members. Like system and collaboration models, organizational units are model elements. Besides, the system and collaboration model can be assigned to organization units: All model elements have a creator and a modifier. Furthermore, organizational units can be responsible for special model elements, such as action items, issues or risks [15] (see Figure 6).

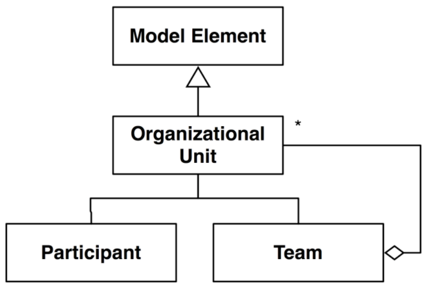


Figure 6: Selected organizational models [3]

5. Modeling Multiple Releases

Modeling multiple releases has special needs as system models keep on evolving from release to release: New system models emerge while existing ones disappear or change. Consider the Multimedia Player (see Section 3) as an example: A new Release, release 3, will introduce TV-shows and support data transfer via bluetooth and WLAN (see Figure 7). Compared to the system model of the previous release 2, the use cases TV-shows and TransferViaWLAN emerge. Similarly, the use case TransferViaUSB disappears as Release 3 offers WLAN and Bluetooth only. That's why the use cases PlaySong and PlayPodcast might change. The modified use case PlaySong in release 3 is called a *refinement* of the use case PlaySong in release 2. To

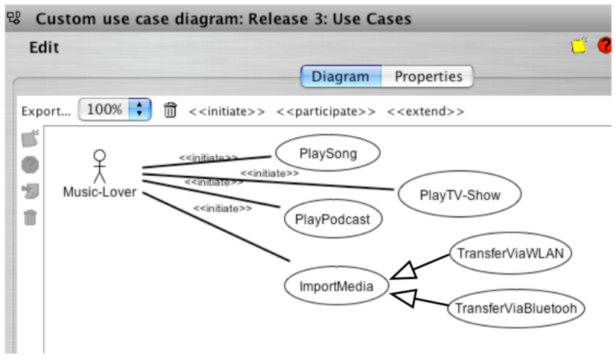


Figure 7: Screenshot of the RAT client with a use case diagram for release 3 of the Multimedia Player

support emerging, disappearing or refining of system model elements is one requirement to model multiple releases.

Traditionally, software configuration management addresses this problem. SYSIPHUS provides mechanisms to develop one system model and track it over time using software configuration management [15]. For modeling multiple releases the use of a software configuration management system is not enough because it only tracks changes along a timeline. It allows reconsidering a model at a particular time, for example at the time of releasing the software. However, to develop multiple releases it should be possible to develop one model per Release simultaneously: While most of the project participants are terminating, e.g. testing and bug fixing, release 3 of the Multimedia Player, an innovation team elaborates ideas for release 4. Both need to develop their own models. New ideas of the innovation team should be isolated from the model of release 3 as it shall be delivered with release 3. In contrast, the innovation team is interested in current changes in release 3. Their model should build upon a realistic system model. Software configuration management does not allow simultaneous development of two independent system models while one is influencing the other.

To address the problem of modeling multiple releases, we introduce release and knowledge nuggets [8] as two new model elements in SYSIPHUS: A *release* represents a version of the system that is made available externally. Releases can be ordered using the setNextRelease()-method: A release is the next release of another, if it is delivered later than the other one. In the example of the Multimedia Player release 2 is the next release of release 1. A release uses one *knowledge nugget* to provide the modeling, collaboration and organizational knowledge of a specific release. Therefore, a knowledge nugget is associated with any

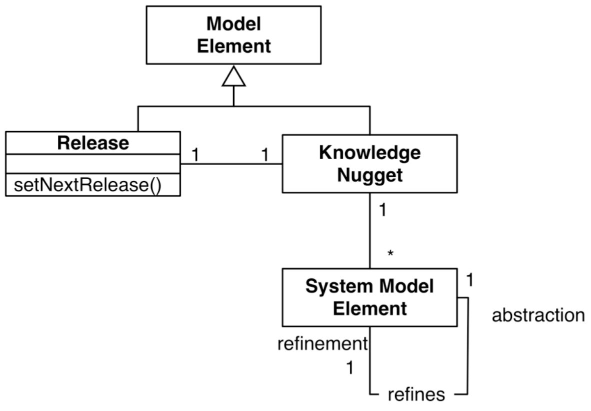


Figure 8: Releases and knowledge nuggets

number of system model elements (see Figure 8). These system model elements represent the system model of the release. Knowledge nuggets use the traceability services of SYSIPHUS to procure the collaboration and organizational models of its system models. A specific system model element however, can only be linked to one knowledge nugget. To map one system model element to several releases, a copy is created. So every knowledge nugget is associated with its own copy of the system model element. By modifying such a copy, the developer refines a system model element of a specific release.

The advantage of this concept is easy refinement of system model elements. However, now these copies have to be managed. The main issue is to assign system model elements to knowledge nuggets of releases at the time of their creation. There are three ways to perform this task: First, the user explicitly selects the release when he creates a model element. As this is a repeating, humdrum activity, the tool analyses the creation context and proposes a possible release for the created model element. The user has to confirm in ambiguous cases only. Second, if the project manager modifies a release plan, the release items of this plan are added to the knowledge nugget of this release. Section 7.2 describes this. Third, knowledge nuggets propagate model elements to knowledge nuggets of later releases. Take the Multimedia Player as an example: After adding the use case PlayPodcast to release 2, its knowledge nugget creates a copy and adds it to release 3.

Aside from new system model elements, the knowledge nugget also propagates modifications to existing ones. That's why a copy is still dependent on the system model element it was created from: Figure 8 expresses this with a refinement association. A refinement link, a specialization of the abstract model link class, implements this association. Knowledge nuggets may propagate changes along refinement links. This happens if the object of change, e.g. an attribute of the refined system model element, has not been changed since it has been copied. This allows propagating changes without overwriting refinements in later releases.

6. Planning Multiple Releases

A release manager plans a release by proposing release items that could be implemented in a specific release. We introduced release plan as a new model element in SYSIPHUS. A release plan aggregates a number of release items. Release items are functional requirements, use cases, features, nonfunctional requirements, issues and bugs (see Figure 9).

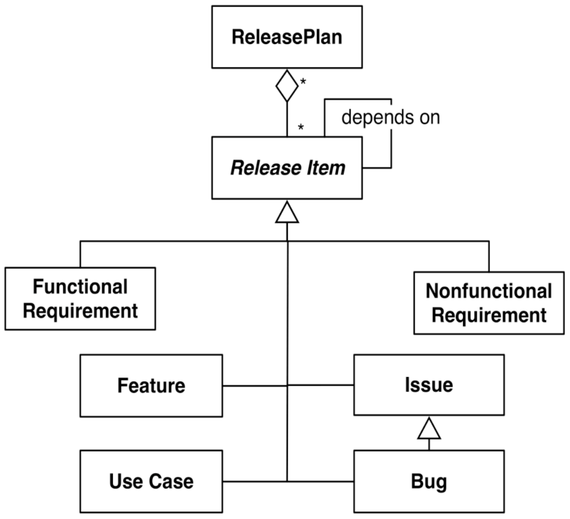
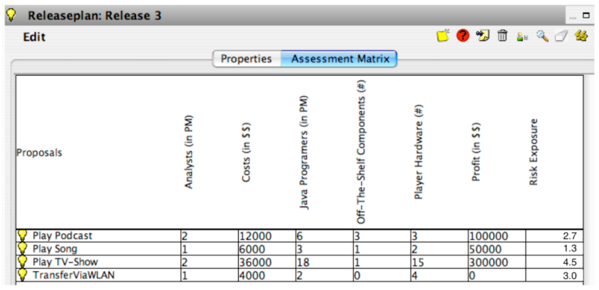


Figure 9: Release plans contain any number of release items

Release items might be dependent on each other: An example for such a dependency is that a release item can only be delivered together with another release item. In release 1 of the Multimedia Player, for instance, the use case PlaySong depends on TransferViaUSB in this way. Analysts should explicitly specify those dependencies when they create release items. Dependent release items cannot be planned in isolation. If a project manager adds release items that depend on other release items, the tool respects the dependencies and adds all necessary release items to the release plan. In the example, the use case TransferViaUSB will be added to the release plan, even if the project manager selects the use case PlaySong only. Those dependencies facilitate the selection of release items: They permit to group single release items to high-level release items. Project managers do not have to consider every single release item.

To provide an informed selection of release items, project managers assess them according to criteria. Sources for criteria range from resources for the release over business numbers to the architecture or models of the system. Figure 10 shows the release plan for release 3 of the Multimedia Player: It contains the new or refined use cases of release 3 as release items, i.e. PlayPodcast, PlaySong, PlayTV-show and TransferViaWLAN. The example considers criteria such as costs, needed number of analysts, java programmers, player hardware and off-the-shelf components as well as expected profit and risk exposure.



Proposals	Analysts (in PM)	Costs (in $$)	Java Programers (in PM)	Off-The-Shelf Components (#)	Player Hardware (#)	Profit (in $$)	Risk Exposure
Play Podcast	2	12000	6	3	3	100000	2.7
Play Song	1	6000	3	1	2	50000	1.3
Play TV-Show	2	36000	18	1	15	300000	4.5
TransferViaWLAN	1	4000	2	0	4	0	3.0

Figure 10: Release plan in the RAT client

These criteria help project managers to select release items for a release plan. Because it is difficult to find an optimal selection [3], the tool allows evaluating alternative release plans to find global solution that respects aspects from different sites. To trade off between alternative release plans, we integrated release planning in the collaboration model. A release has a special release issue: Which release plan shall be implemented in this release? Project managers can elaborate several alternative release plans as a proposal for this release issue (see Figure 11).

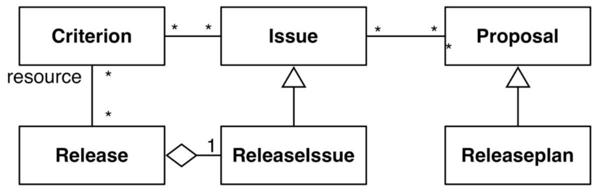


Figure 11: Alternative release plans

To support the selection of a release plan, the alternatives are assessed according to criteria (see Figure 11). The tool calculates criteria assessments for every release plan. This calculation is based on the assessment of the release items. Assessments for release items often can be summed up to assessments for the release plan. However, there are criteria that use other calculation rules, such as risk exposure that can be summarized using a geometric series. Another example is a criterion denoting the number of release items with a high priority. Therefore, we introduced calculated criteria. A calculated criterion is a criterion whose value can be calculated using a formula. A formula consists of a calculation rule, such as sum or geometric series. To perform this calculation, a formula uses parameters as input. A parameter is either a criterion or a formula (see Figure 12). In SYSIPHUS a proposal provides its assessment of a criterion (see Figure 5). According to this, release plans calculate values of calculated criteria. For this purpose, the release plans ask their associated release items for assessment regarding a criterion of the formula. Then, it uses the formula to calculate the assessment of its calculated criterion.

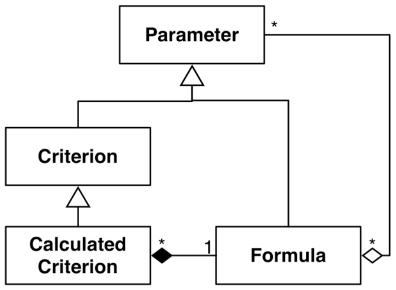
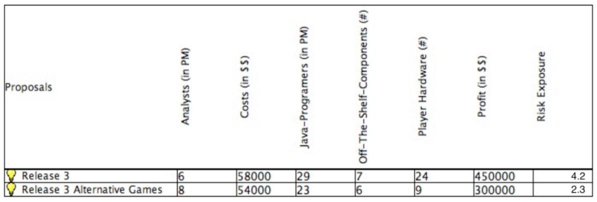


Figure 12: Calculation of assessments for release plans

Figure 13 shows a screenshot of a release planning view for release 3 of the Multimedia Player comparing two alternative release plans: Alternatively to the plan presented in Figure 10, the project manager considers to offer games instead of TV-shows. The assessments for the criteria in Figure 13 are calculated as described above. The project manager can create and add any criteria to the release. Nevertheless, he should respect the resources of the release as criteria (see Figure 11). For this reason, the tool always adds resources, which constrain a release, to the release issue as criterion.



Proposals	Analysts (in PM)	Costs (in $$)	Java–Programers (in PM)	Off–The–Shelf–Components (#)	Player Hardware (#)	Profit (in $$)	Risk Exposure
Release 3	6	58000	29	7	24	450000	4.2
Release 3 Alternative Games	8	54000	23	6	9	300000	2.3

Figure 13: Evaluating alternative release plans in the RAT client

7. Awareness

Awareness denotes being conscious of other project participants' changes and actions at different sites to avoid misunderstanding and to increase trust. [3] Release modeling and planning introduces two new types of changes of which other project participants need to be aware: Changes to release models and changes to release plans.

7.1. Awareness of Changes to Release Models

Changes to release models, e.g. adding a use case to the functional model of a specific release, arise from current development activities. Two groups of project participants are interested in those changes: Developers and project managers.

Developers might be interested in changes concerning special model elements they have created, modified or which have an impact on their work. SYSIPHUS supports them by tracking activities, modifications and authors of changes of all model elements. [3] The RAT client, for example, uses this information to display the date and author of the last modification to model elements. The software configuration management of SYSIPHUS allows reconsidering the history of model elements.

Project managers might be interested in refined, emerged or disappeared release items of release models that affect release plans. Project managers should be aware of these changes and respect them in release plans as they might need resources and influence the desired outcome. The proposed approach addresses this by applying changes in the release model to its plan: The tool adds refined or emerged release items to the release plan; similarly, it removes disappeared items from the plan. Consider release 3 of the Multimedia Player as an example: A developer refines the use case PlaySong as USB transfer is not supported any more. As a consequence the tool adds it to the release plan of the release 3. This mechanism guarantees that the plan is not a crystal ball, but reflects reality.

7.2. Awareness of Changes to Release Plans

Section 7.1 describes a mechanism to apply changes from release models to their plans. SYSIPHUS supports project managers to be aware of these changes to the release plan by tracking changes [3]: The RAT client, for example, presents the latest changes together with their authors. This mechanism helps project managers as long as they observe release plans regularly. Otherwise SYSIPHUS provides an email notification: Users can register to obtain notifications of the meta model [3]. Project managers, for instance, can register for changes to release items of a release plan. If this release plan changes, e.g. due to a emerging release item in a release model, subscribed project managers receive an email.

Aside from project managers, developers are interested in changes to release plans as they implement them. To increase collaboration between project managers and developers, the proposed approach applies changes in release plans to the release model. Whenever project managers change release plans or decide for another release plan alternative, the tool updates the release model: It removes release items from the release model, which are part of the model, but not of the release plan. Similarly new release items in the plan will be added to the model, if they are still not part of the model. The tool replaces refinements of system model elements in the model with those of the release plan. By applying the latest changes of the project manager to the system model, the approach of this paper increases developers' awareness of changes in the release plan.

8. Traceability

Release planning has to deal with change: Release plans change when the requirements change, open issues are identified or bugs are detected. Release plans also change according to organizational or economic conditions: If an important participant leaves the project ahead of time, a project manager has to decide how to proceed. The unified meta model for system, collaboration and organizational models, consisting of the abstract classes model element and model link allows traceability to update models, identify change impact and detect dependencies from resources. [3]

Explicit links allow a *model update*: The explicit link from use cases to classes, for instance, allows updating the class model, if the use case model changes [3]. If the project manager of the Multimedia Player decides to introduce games instead of TV-shows in release 3, the tool uses this link to add all classes to the model of release 3 that are necessary to implement games. Similarly, the tool removes all classes of the use case PlayTV-show from the model of release 3, which are only needed for the use case PlayTV-show. In Figure 14 a screenshot of RAT shows a class diagram with the supported products of release 3 after this change.

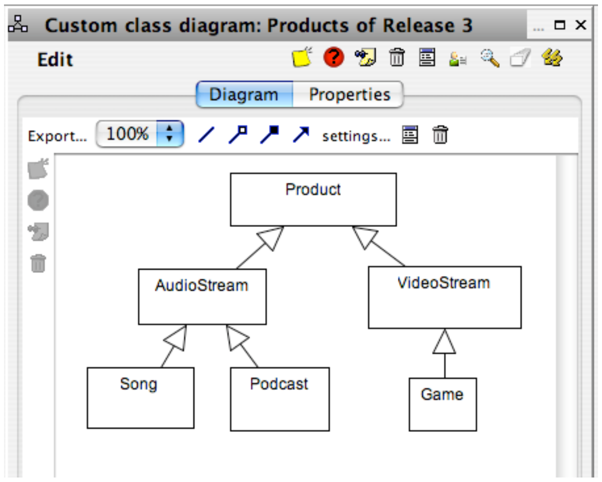


Figure 14: Class diagram for release 3 of the Multimedia Player in RAT

For release planning SYSIPHUS supports tracing *change impact*: If a bug is identified in a class, the project manager can trace back to the use cases and see which use cases are affected [3]. This helps him to estimate the impact of a bug on the release.

Project managers can observe *dependencies* from resources, for example certain participants: By annotating model elements and assigning participants to annotations, project managers can trace participants to model elements [3]. For example, SYSIPHUS can show him all issues or action items which a participant is assigned to. So, project managers can evaluate the consequences, if participants will not be available any more.

9. Conclusion and Future Work

The proposed tool is based on SYSIPHUS that emphasizes system, collaboration and organizational models equally. We introduce knowledge nuggets to address the problem of release planning and modeling for innovative, distributed software projects. In these projects requirements often are vague at project startup and are specified slowly during development. The proposed tool deals with evolving requirements: If developers change a specific model at any site, e.g. by adding, removing or refining requirements, the tool updates the release plans. The tool increases the project managers' awareness of changes in the model and informs him about changes by sending an email. Providing awareness is a key to reduce misunderstandings and redundant work in distributed projects [3]. Then, project managers might review the plan and decide how to deal with changes. The tool allows an informed decision by providing traceability, alternative release plans and assessment of criteria. Traceability shows change impact and dependencies from resources. Alternative release plans are a mechanism to evaluate possible sets of requirements. Participants of a distributed release-planning meeting could elaborate alternatives synchronously during the meeting. These might be used to reflect local decisions and support project managers to find a global solution respecting local ones. The tool calculates assessments of user-defined criteria for every release plan alternative. This calculation is based on the criteria assessment of release items. The assessment of release items and release plans eliminates decisions that are based on instinct. If project managers revise their decision, the tool updates the release models of the system: It adds and removes release items to the model respecting their dependencies. Dependency management allows grouping release items and reduces the complexity of their selection by shortening the number of release items that must be considered. Using traceability, the tool also applies changes in the release model resulting from changed release items to the class models of the release.

Knowledge nuggets, releases and release plans have been implemented in SYSIPHUS as proposed. SYSIPHUS is used, evaluated and evolved successive since 2000 within student projects and consulting projects for industry. Our student projects provide a realistic environment: A real client proposes an actual problem. The projects involve up to 100 participants in Germany, USA and New Zealand [3]. We plan to evaluate the concepts presented in the paper in our next student projects.

References

[1] A. J. Bagnall, V. J. Rayward-Smith, I. M. Whittley: "The Next Release Problem", In: "*Information and Software Technology*", 43(14), pp. 883-890, 2001.

[2] Bernd Bruegge, Allen Dutoit: *Object Oriented Software Engineering - Using UML, Patterns and Java.* Prentice Hall, Upper Saddle River (NJ), 2nd edition, 2003.

[3] Bernd, Bruegge, Allen Dutoit, Timo Wolf: "Sysiphus: Enabling informal collaboration", In: *"Proceedings of the 1st International Conference on Global Software Engineering*", Costão do Santinho, Florianópolis, Brazil, October, 2006.

[4] Allen H. Dutoit, Raymond McCall, Ivan Mistrik, Barbara Paech: "*Rationale Management in Software Engineering*". Springer, Berlin, 2006.

[5] Pär Carlshamre: "Release Planning in Market-driven Software Product Development - Provoking an Understanding", In: "*Requirements Engineering Journal"*, 7(3), pp.139-151, London, September 2002.

[6] D. Greer: "Decision Support for Planning Software Evolution with Risk Management", In: "*Proc. 16th International Conference on Software Engineering and Knowledge Engineering"*, pp. 503-507, June, 2004.

[7] D. Greer, G. Ruhe: "Software Release Planning: An Evolutionary and Iterative Approach." In: "*Information and Software Technology*", Volume 46, pp. 243-253, 2004.

[8] Korbinian Herrmann, Bernd Bruegge: „Visualization of Release Planing." In: *"Proceedings of 1st International Workshop on Requirements Engineering Visualization" (REV'06), IEEE Requirements Engineering*, Minneapolis-St.Paul, Minnesota, 2006.

[9] MicroTool-Homepage: http://www.microtool.de/suite/de/releaseplanung.asp [11.05.2007]

[10] J. Momoh, G. Ruhe: "Release planning process improvement - an industrial case study", In: "*Software Process: Improvement and Practice"*, Volume 11, Issue 3, pp. 295-307, 2006.

[11] ReleasePlanner-Homepage: http://www.releaseplanner.com [11.05.2006]

[12] O. Saliu, G. Ruhe: "Supporting software release planning decisions for evolving systems", In: "*Proceedings of 29th IEEE/NASA software engineering workshop"*, Greenbelt, MD, USA, 2005.

[13] Sysiphus-Homepage: http://sysiphus.in.tum.de [11.05.2007]

[14] VersionOne-Homepage: http://www.versionone.net/product_planning.asp [11.05.2006]

[15] Timo Wolf: "Sysiphus: Modellbasierte Kollaboration in Software-Entwicklungsprojekten", In: "*Objektspektrum"*, Volume 2, pp. 30-35, Troisdorf, MarchApril 2007.

The Challenges of Distributed Software Engineering and Requirements Engineering: Results of an Online Survey

Timea Illes-Seifert[1], Andrea Herrmann[1], Michael Geisser[2], Tobias Hildenbrand[2]
[1] *Institut für Informatik, Neuenheimer Feld 326, 69120 Heidelberg, Germany*
{illes;herrmann}@informatik.uni-heidelberg.de
[2] *Lehrstuhl für ABWL und Wirtschaftsinformatik, Schloss, 68131 Mannheim, Germany*
{geisser;hildenbrand}@uni-mannheim.de

Abstract

Growing globalization and increasing complexity of software lead to international and national collaboration of geographically distributed organizations, sites and persons. Therefore, it becomes more important to understand and to know how to optimize distributed software development. Thus, we performed a survey among professionals on their experiences with distributed software development. We present an evaluation of 744 questionnaires, with a special focus on requirements engineering. The survey results show that a variety of human and process-related aspects are important for distributed software development. They furthermore emphasize the importance of communication in requirements engineering: Communication, particularly face-to-face meetings, represents the most frequently mentioned solution to diverse problems. Similar results were found before, but this survey supports them with a high quantity of data.

1. Introduction

The trend towards sub-contracting, outsourcing, and off-shoring, as well as the collaboration with partner organizations or within an organization at different locations (nationally and internationally) requires the use of knowledge and resources distributed over multiple locations. Communication, as the process of knowledge exchange, is therefore an important issue for software development teams [2] - even when they are not distributed: „Software Engineering is inherently a team-based activity“ [1] and thus implies knowledge exchange among its members. In the case of distributed projects, communication becomes even more important [7], [8], [19]. Existing research indicates that means of communication, such as phones, mobile devices, email, or video conferencing equipment cannot fully substitute face-to-face meetings and demand for instance communication by traveling [4] or “get to know” meetings [6]. Although tool support and processes which support collaboration cannot guarantee a good software engineering result, they are considered to be necessary prerequisites. Requirements on such tools supporting distributed software engineering are discussed in [8] and requirements on distributed processes in [3]. A tool for requirements prioritization by “non-co-located experts” is presented in [13] and a process for distributed requirements prioritization in [21]. First studies investigated lessons learned from distributed software development [20] and distributed design [7].

As their conclusions however are based only on a few cases, we performed a quantitative online survey among software engineering professionals with the goal to investigate the state of the practice in distributed projects, including distributed requirements engineering. Particularly, we used an online survey in order to reach a high number of participants and consequently to derive statistically significant results.

High participation in our survey indicates the practical significance of distributed software development (744 participants within 3 weeks). Moreover, the high degree of experience with distributed software projects among the participants underlines the study’s representativeness.

In this paper, we present the major results of the online survey. The remainder of the paper is organized as follows: Section 2 provides a description of the survey, while section 3 presents characteristics of the participants, of the distributed processes and of the tool usage as indicated by the respondents. Section 4 describes the analyses of participants´ responses related to challenges of distributed software development as well as issues specific to distributed requirements engineering (RE). Additionally, successful solutions to the issues mentioned by the participants are presented. Overall conclusions and future work are provided in Section 5.

2. Methodology and Study Design

Distributed Software Development. In our study, we define *distributed software development* as follows: "*All or at least some participants of a software project predominantly use distributed technologies for team communication (e.g. because this is not possible otherwise due to geographical distance)*" [12].

Questionnaire Design. We designed our survey questionnaire and the categories for the coding of answers to open questions by applying the MOQARE/misuse case principle [10], [11]. We first defined important quality goals of each – even intermediate – product, i.e. of the requirements specification, design, code, and test results during distributed software development. This quality was measured by quality attributes which were specific for each product. As the quality goal of the process we defined the efficiency in the creation of these products. Then, we identified misuse cases, which can possibly happen during the process of distributed software development and which endanger the goals mentioned above. Misuse cases describe scenarios which must be avoided. Discussing such unwanted events and the countermeasures that can detect, prevent or mitigate them, usually helps to complement requirements. In the next step, we identified such countermeasures which here were requirements for processes and tools used in the development of distributed software. We identified misuse cases and countermeasures for the different phases of the software development: for requirements engineering, architectural design, coding and testing. An example is the misuse case "The requirements specification is ambiguous because different terminologies and notations are used." Important countermeasures for this misuse case would be to use a glossary and to define a common notation. Many misuse cases could occur the same way in every activity and were classified as "general problems". For two reasons, we did not include our 137 misuse cases in the questionnaire. Firstly, this would demand too much time of the survey participants to comment on all of them, and secondly, pre-defining a list of misuse cases would focus the answers on these particular ones, without guaranteeing that the list contains the most relevant ones as experienced by the participants. Instead, the misuse cases were the basis for coding the answers to the open questions during data analysis.

The questionnaire consists of two parts. The first part contains questions on the respondent's experience with distributed software development. Particularly, we asked about the roles of the respondent, the phases which have been performed in a distributed way, as well as the technology used for communication and information sharing in distributed projects. The analysis of the respondent's answers to questions covered by the first part of the questionnaire is presented in section 3. The second part of the questionnaire addresses problems that occur in distributed projects, their causes and their solutions. This second part consists of four sets of questions. In the first set, respondents were asked about general problems in distributed projects and their solutions (here, we proposed those nine misuse cases which apply to general problems), whereas the other three sets of the questionnaire asked open questions concerning problems and solutions specific to distributed requirements engineering, software design and coding as well as software testing. An analysis of the respondents´ answers to questions covered by the first set is presented in Section 4.1, whereas comments, misuse cases and countermeasures concerning distributed requirements engineering are presented in Section 4.2.

The resulting questionnaire was thoroughly reviewed and tailored by the authors before being published online. Criteria for reviewing were above all the comprehensibility of the questions. Another criterion was the time needed for answering the questionnaire. Since respondents are not willing to deal with lengthy questionnaires [17], we aimed at developing a questionnaire, which does not take longer than 20 minutes to be completed.

Data Collection. The final version of the survey was published online for three weeks. In order to attract many participants, we promoted the questionnaire by posting an online advertisement in the news ticker of a popular German computer journal. Additionally, we addressed the participants on the mailing list of events organized by the MFG (Media and Film Association) Baden-Württemberg, a centre of excellence for information technology and media in the southwest of Germany [16].

Data Analysis. Before performing analyses, we validated the data, analyzing the responses with respect to validity and consistency as recommended in [18]. E.g. 24 participants indicated having no experience with distributed development projects, thus, we did not consider their responses in our further analysis. Finally, there were 744 valid questionnaires. The mean time for completing the questionnaire finally was 14 minutes.

After data validation, we coded the answers [18]. Thus, answers to open questions were categorized in order to be analyzed in further steps. In questions concerning the technology used to support distributed software development, we had proposed several alternatives including email and chat. However, the participants also had the opportunity to add other technologies not mentioned in the list. Some of the respondents indicated special software packages, requiring that we had to code the information and to categorize the answers by assigning the particular software solutions to a particular underlying technology.

3. Results

In the following, general characteristics of the participants, of the distributed processes and of the tool usage as indicated by the respondents are presented.

3.1. Participants Characteristics

Experience of the Participants. The participants had worked in an average of 7.5 distributed projects. This shows the high qualification of the participants, and also that distributed software development is neither an exotic, nor a newly emerging phenomenon.

Roles of the Participants. The most frequent role the participants had taken in distributed projects was the one of the developer (68%) and the software architect (57%). 39% stated they were project manager, 16% requirements engineer, 29% tester and 7% in other roles. (Double and multiple roles were indicated frequently.)

Application Domains of the Software. The participants were asked about the business domain of their customers (multiple answers were possible). Most frequently, respondents stated "software" (48%) followed by the technical sector (42% including mechanical engineering, chemistry, electrical engineering, telecommunication, and transport) and service (39% covering education, consulting, IT services). The rest were commercial sector (banking, insurance) (23%), public sector (administration, government) (19%) and others (14%).

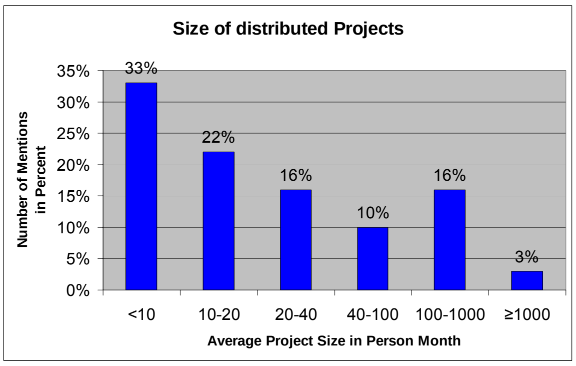
3.2. Characteristics of Distributed Processes

Size of Organizations. Software is being developed in distributed projects in big organizations as well as in small and medium sized organizations: 34% of these projects took place in organizations with more than 10.000 employees, and 38% in such with less than 100. The rest is distributed among organizations with 1000-10.000 employees (13%) and 100-999 (15%).

Size of Projects. 33% of the distributed projects had a volume of less than 10 person months and another 22% from 10 to 20. Figure 1 illustrates the average size of distributed projects as indicated by the respondents.

An average of 94 persons per project communicated via the distributed technology, and the average number of project team members was 84. It can be concluded that the distributed technology involved persons in the communication who were not project team members. On the other hand, 18% of the participants did not know the number of persons involved. This high number indicates that the distributed communication leaves some persons without an overview or "awareness" [9] about the members of the team and their activities.

Figure 1. Average Project Size in Person Month



Project Phases and Roles. The project phase which had been done in a distributed way most often was implementation (92% of the participants), followed by testing (73%) and architectural design (62%). Requirements analysis (38%) and operation as well as maintenance (46%) were less frequent. This distribution among project phases is also reflected by the role distribution of the participants (see preceding section) and when being asked which roles did use the distributed technology. We want to point out that 27% of the respondents indicate that later users of the system were also involved in the development project using distributed technology for communication and information exchange.

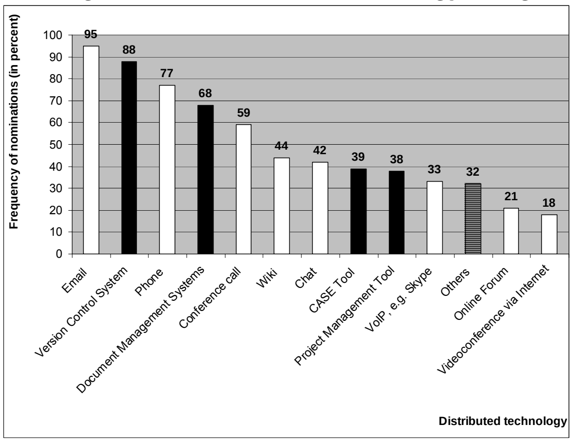
3.3. Tool Characteristics

Distributed Communication. Participants of the survey were asked to indicate which kinds of distributed technology they use for distributed communication. Email seems to be the most important tool for communication. Almost 95% of the participants indicate to use those means for asynchronous communication. The most important synchronous technologies are the phone and the conference call. 77% of the respondents indicate to use phone and 59% indicate to use conference calls in distributed projects. Other technologies used comprise video conferencing (not via internet) and remote desktops.

Distributed Information Exchange. Participants of the survey were also asked to indicate which kinds of distributed technology they use for distributed information exchange. Version control systems and document management systems referred to as the most frequently used technologies for information sharing. CASE tools and project management tools are only used by about 40% of the respondents to exchange information. Another information platform mentioned by respondents represents problem/defect management reports.

Figure 2 summarizes the results of the survey study with respect to the frequency of mentions regarding communication (white bars) and with respect to information exchange using a certain distributed technology (black bars). The finding, that email, telephone and file sharing are the most frequently used tools is consistent with results of other studies [6],[14].

Figure 2. Distributed Technology Usage



4. Analysis of Participants´ Comments, Misuse Cases and Countermeasures

In this section, we present challenges of distributed software projects and particular issues in requirements engineering drawn from our survey.

4.1. Challenges of Distributed Development

In comments and open questions concerning misuse cases, respondents mention mainly five types of challenges concerning distributed development: process barriers, cultural barriers, domain specific barriers, technical barriers and communication barriers. Figure 3 visualizes these barriers and their corresponding specific facets by means of a fishbone diagram. The numbers in parentheses represent the number of mentions.

Process barriers are the most frequently mentioned barriers in distributed software development. 10 respondents indicate as a major problem that documented processes are not actually implemented and that responsibilities are not clearly defined (mentioned 9 times). Reasons for unclear responsibilities as mentioned by the respondents are, e.g. frequently changing responsibilities or the lack of a coordinator role. Other important process barriers represent enhanced communication needs (9) and inappropriate processes (8). A main reason for increased communication is reported in cases with incomplete documentation. Inadequate processes mainly result from the use of “standard” processes which are not adjusted to the distributed context. A special case of inadequate processes represent inflexible processes (7). Respondents emphasize the problem of rigid processes, where changes are very slowly propagated. Finally, other process barriers reported by the respondents include undefined requirements concerning the tools and infrastructure to be used, resulting in inappropriate tools, missing commitment from the management, above all concerning quality assurance activities related to distributed processes and undefined processes.

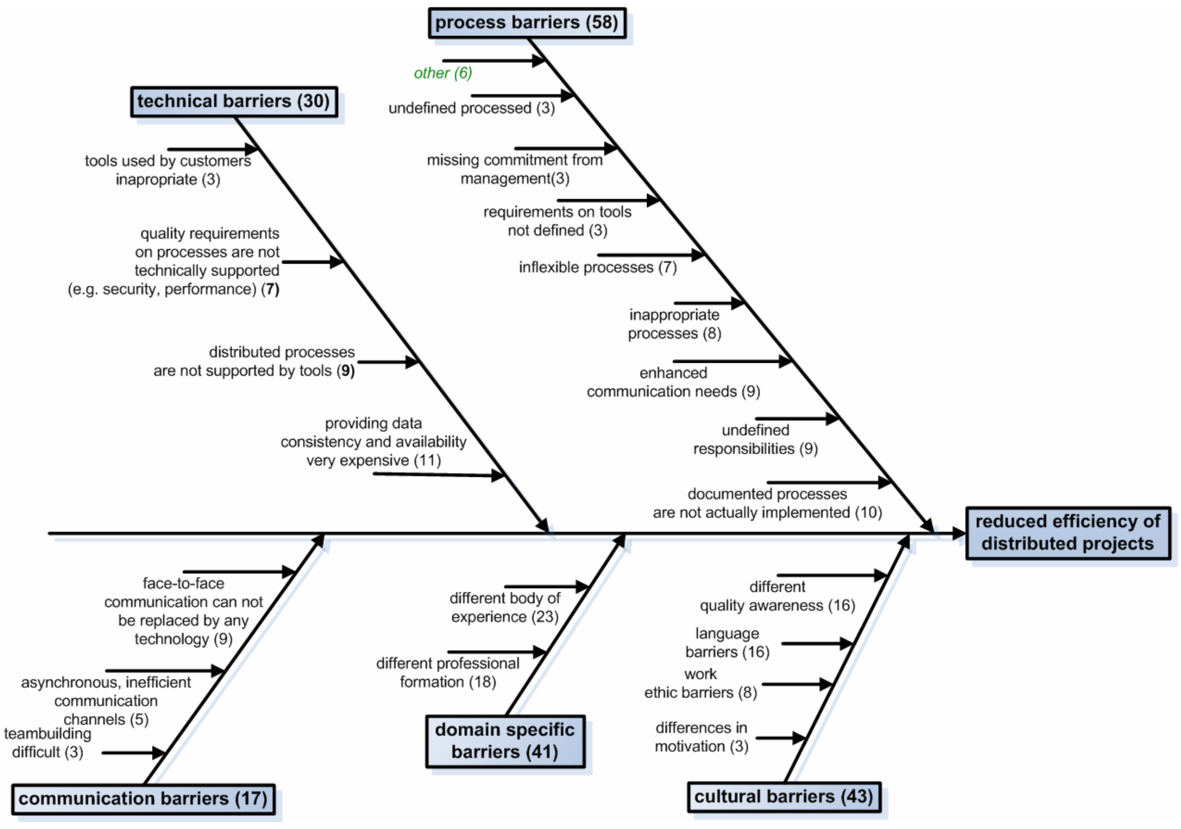
Main facets of *cultural barriers* mentioned by the respondents are not only differences concerning the language (mentioned 16 times). Differences in the awareness of quality (16) or work ethic barriers (8) influence distributed projects, too. One respondent highlights the problem that for some cultural groups it is difficult to express disagreement to customers. Consequently, “nice-to-have” features as well as key features are treated equally, resulting in requirements without priorities. Another participant reports that countries differ in the work ethic with respect to the accuracy of the work as well as to the ability to improvise.

Domain specific barriers mainly subsume differences concerning experience (mentioned 23 times) and the professional formation (18) of distributed teams. Respondents report four kinds of experiences missing in distributed projects: experience in general, experience concerning distributed projects, domain specific experience and experience with specific tools.

Technical barriers also influence the efficiency of a distributed project. Respondents report that information in form of data is often distributed. Providing consistency and availability of the data are the most important technical barriers (mentioned 11 times). Another technical barrier is that tools do not support distributed processes (9) and quality requirements on processes (7). Particularly lacks in security and performance of the tools often prevent their (efficient) usage. Another technical barrier is that tools used by customers are inappropriate and do not integrate well with tools used within the organization (3).

Missing face-to-face communication is a specific *communication barrier* and it is seen as indispensable even when technological support for synchronous or asynchronous communication is available (mentioned 9 times). One respondent indicates that technology does not replace a convivial evening having a “glass of wine or beer” together. The use of asynchronous, inefficient communication channels represents an often mentioned communication barrier (5). Additionally, respondents also indicate that distributed team building to facilitate communication is a very difficult task (3).

Figure 3. Challenges of distributed development



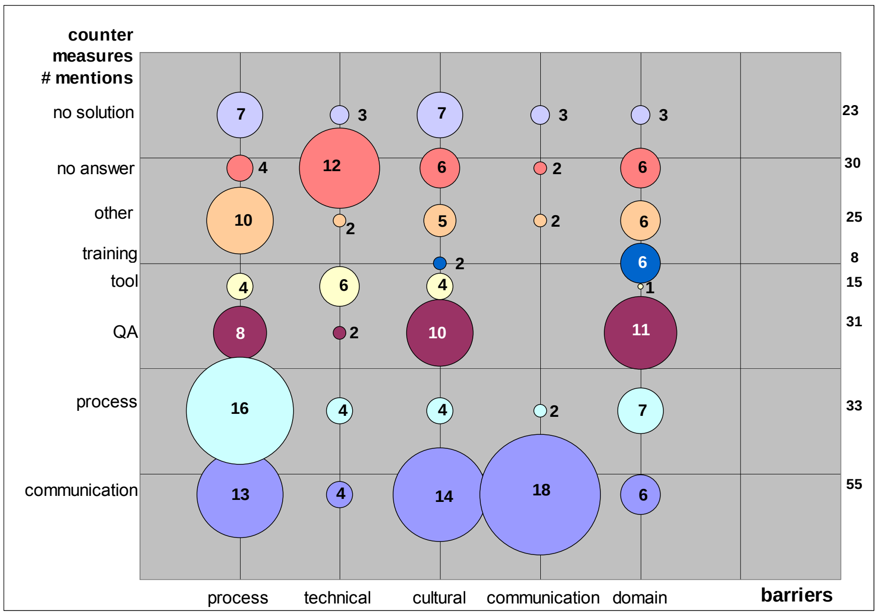
process barriers (58)
other (6)
undefined processed (3)
missing commitment from management(3)
requirements on tools not defined (3)
inflexible processes (7)
inappropriate processes (8)
enhanced communication needs (9)
undefined responsibilities (9)
documented processes are not actually implemented (10)
technical barriers (30)
tools used by customers inapropriate (3)
quality requirements on processes are not technically supported (e.g. security, performance) (7)
distributed processes are not supported by tools (9)
providing data consistency and availability very expensive (11)
reduced efficiency of distributed projects
face-to-face communication can not be replaced by any technology (9)
asynchronous, inefficient communication channels (5)
teambuilding difficult (3)
communication barriers (17)
different body of experience (23)
different professional formation (18)
domain specific barriers (41)
different quality awareness (16)
language barriers (16)
work ethic barriers (8)
differences in motivation (3)
cultural barriers (43)

4.2. Successful Countermeasures

We asked the participants about countermeasures to problems occurring during distributed software engineering, which had successfully been applied. For 136 of the 189 problems, the participants indicated countermeasures. In 30 cases, the participants did not give any answer and in further 23 cases the participants explicitly indicated that there was no successful countermeasure.

The solutions to the barriers presented in Section 4.1 can also be grouped into 5 main categories: communication, process, quality assurance (QA), tool and training. An additional category "other" subsumes solutions which can not be assigned to any of the categories mentioned above. Figure 4 summarizes the responses of the respondents and assigns to each barrier the absolute number of solutions indicated by the participants per category. The most important countermeasure to barriers in distributed software development is communication. Above all, intensifying communication is indicated as an important countermeasure to almost all barriers (mentioned 55 times). Above all, communication by email and face-to-face communication were indicated as successful countermeasures in this category.

Figure 4. Countermeasures per barriers – absolute number of indications



Another important group of countermeasures deals with *process* aspects (mentioned 33 times). Within these mentions, process improvements, a clear definition of responsibilities as well as the definition of process standards were indicated as successful countermeasures in this category. Additionally, the definition of a flexible and iterative development process was also mentioned.

Intensifying quality assurance activities represents another group of countermeasures (mentioned 31 times). Above all, the definition of standards and the performance of more frequent reviews and audits were indicated as successful countermeasures in this category. The definition of standards subsumes the definition of a standard terminology and of a standard language as a countermeasure to communication and domain specific barriers. Additionally, the definition of standard templates has proven of value to overcome domain specific barriers. Finally, intensifying reviewing activities is also a countermeasure to domain specific barriers.

Noticeable is that for about half of all technical problems the participants could not indicate successful solutions.

4.3. Challenges of Distributed Requirements Engineering

In addition to the general problems discussed in the preceding section, in another part of the questionnaire we asked whether there were problems specific to distributed requirements engineering. 58% of the participants answered that they had no problems specific to this phase and to distributed software development. 17% answered that there were such problems, but they did not know them, and 25% said there were problems, and these all together listed 122 of such problems.

We wondered how it was possible that 58% reported no specific problems. (As we will show later, in fact the reported problems are specific to distributed requirements engineering, as was the intention of this question.) To find out why this proportion is so high, we first examined to what degree the person's own role influenced his/her answer. Of all those participants who had the role of requirements engineer in the distributed projects, 33% reported detailed problems, 12% said there were such problems, but they did not know which, and 55% said there were none which were specific. So, requirements engineers reported about RE problems only slightly more often than the average participant. Three further explanations for the low percentages of reported specific problems, which probably all are valid to a certain degree, can be: Many of the RE problems observed during distributed software development would have happened likewise in non-distributed projects and therefore were not reported here. Or the participants did not want to answer this question, either because the questionnaire seemed too long to them or because they did not want to give too detailed confidential information about problems.

We also wondered whether many participants did not report RE problems as this phase was not done in a distributed way. 146 participants reported that the RE phase was distributed. For 34% of them, problems were reported, 17% experienced problems without knowing them and 49% seem to have had none. Amazingly, only 57% of those participants who reported RE problems which are specific to distributed software development, also had reported that the RE phase had been performed in a distributed way. These replies are inconsistent. It is possible that the question "Which phase was performed in a distributed way?" was misunderstood by participants, maybe because practitioners do not use the concept of phase. We do not think that the question about problems, which are specific to distributed RE, was misunderstood. Evidence that the participants did understand the question correctly is the fact that such problems which are specific to RE, but not to distributed RE, were rarely reported. Those were found in other studies on RE problems, as in [19]: understanding the users´ needs, conflicts among different customers, how to prioritize requirements, requirements changes.

As was described in Section 2, we asked to name up to three problems (without pre-defined answers) in the part of the questionnaire concerning RE, and in open questions we asked for causes of the problems and for successful countermeasures. In addition to these three problems, further comments concerning RE could be given.

To code the answers to the open questions, we proceeded as follows: We defined the goal of requirements engineering to be the quality of the requirements specification in terms of the quality attributes defined by the IEEE Standard 830-1998 [15] (correctness, unambiguousness, completeness, consistency, verifiability, ranking according to importance and/ or stability, modifiability, traceability) and the efficiency of the specification process. These were the quality criteria we used for the coding of the reported problems.

Each problem reported in the survey was assigned to the quality criterion it endangered. In a second dimension, the reported problems were coded according to the cause of the quality problem observed. These causes were coded according to the types of barriers in Figure 3.

In our MOQARE analysis preceding this survey, we had identified 53 potential misuse cases. Misuse cases combine a cause with a resulting quality problem. (In fact, a misuse case includes much more information, but for our present purpose this simplification is useful.) The analysis of the reported requirements engineering problems led to 13 further misuse cases.

Of the 122 problems which the participants reported, 7 could not be coded, because of vague wording. 47 were related to ambiguity of the requirements specification, 44 to the efficiency of the process and 12 to the completeness of the specification. Only five were related to modifiability, three to correctness, two to consistency, one to verifiability, one to prioritization and none to traceability.

The participants named 19 out of the 66 misuse cases more than once. In the following, the ambiguity and efficiency misuse cases are discussed in more detail, because they were clearly named most often. Table 1 illustrates which type of barrier is observed in which context. Significant differences can be observed. For instance, communication barriers play a more important role in RE than in software development in general, and such communication barriers rather lead to inefficient processes than to ambiguous specifications. The ambiguous specification was mostly (at 66%) attributed to domain specific barriers, which were less important for process efficiency, but highly relevant in RE overall. Such domain specific barriers can be lack of technical knowledge as well as domain knowledge. Technical barriers played an even smaller role in RE than in software development in general. Four times, email and phone were mentioned (both together), but because the problem did not spring from the technology itself, we did not count them as technical barriers. Rather these four answers stated that face-to-face communication cannot be replaced by any technology, so we assigned them to communication barriers.

Problem cause	(a)	(b)	(c)	(d)
Communicatio n barrier	9%	27%	11%	41%
Domain specific barrier	22%	33%	66%	5%
Cultural b.	23%	16%	21%	18%
Technical b.	16%	3%	0%	5%
Process b.	31%	21%	2%	32%

Tab. 1: Problem causes: (a) in distributed software development in general (data from Figure 3 for comparison); (b) in distributed RE; (c) in distributed RE and leading to ambiguity of the requirements specification or (d) leading to an inefficient RE process (columns add to 100%, i.e. percentage tells the ratio of each barrier within each context)

In addition to the coarse-grained statistics in Table 1, in the following some chosen detail information further illustrates the nature of problems in distributed RE. In the context of ambiguous specifications, half of the cultural barriers were of the type "language barriers"; this is more than in software development in general (compare to Figure 3). In distributed RE, 42% of the domain specific barriers meant different terminology or notation of requirements.

In the context of efficiency of the specification process out of the 18 mentions of communication barriers, 5 stated that face-to-face communication cannot be replaced by indirect respectively distributed communication. The other three sub-types of communication barriers with three answers each were: not enough communication, time zones, and asynchronism of the communication. Among the 14 mentions of process barriers, the most frequent ones were undefined responsibilities (5), high numbers of stakeholders as sources of requirements (4), and suitability of processes (3). Out of 8 mentions of cultural barriers, language barriers were mentioned 4 times.

4.4. Successful Countermeasures for Distributed Requirements Engineering

The survey participants were also asked about successfully applied countermeasures for the indicated misuse cases/problems. For only 37 of the 122 problems, such countermeasures were named (as there were often several countermeasures per misuse case, this made 45 countermeasures in all). In seven further cases, the answer explicitly was that there was no successful countermeasure (so far).

As can be seen in Figure 5, the most frequently proposed countermeasures were communication measures (mentioned 16 times) or, more specifically, face-to-face communication (12). This sums up to 28 out of 45 (i.e., 62%). It was proposed to communicate more often, immediately as a question arises, according to formal rules, using a tool (a wiki in this case) and a common terminology. It is remarkable that in distributed development in general (see section 4.1), face-to-face communication was explicitly named only 4 times out of 55 communication measures, i.e. at 7%, and not at 43% as in RE.

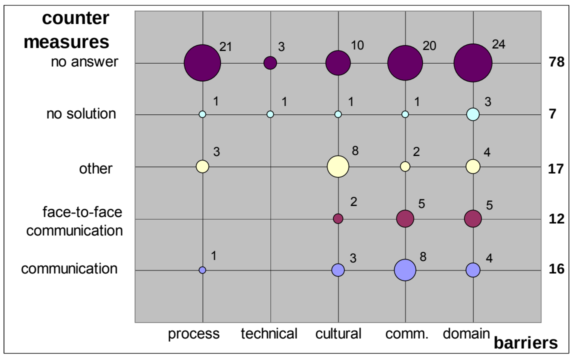


Figure 5. Countermeasures for barriers in distributed RE: numbers of mentions ("Comm." = "communication")

The other (17) countermeasures were: Quality Assurance (here: reviews and inspections) reduced ambiguity of the specifications when it is due to language problems, double work done due to unclear responsibilities, and general human communication problems. Training (here: coaching and workshops) helped against culturally caused misunderstandings and the lack of qualification which had led to incomplete requirements (both domain specific barriers). Three times working more was named.

Specific RE countermeasures were proposed for the ambiguity of requirements specifications which is due to different terminology or notation. These were: example requirements, a glossary, early test specifications, standardization of formats, and the definition of minimum standards for documents. When team members differ in working speed, they must be pushed or their work passed on to faster groups. Conflicts among a multitude of stakeholders are handled by the project manager, e.g. by defining goals which are shared by all stakeholders. Other process barriers were tackled by process improvement, i.e. by a formal change process (2) and regular "polling" (1).

Tools were named three times as countermeasures to communication or quality problems. These tools were: video conferencing, VoIP and Wiki. There were two countermeasures which mention email, but one was counted among the communication countermeasures (because it said to communicate via email and phone, so the advice was to communicate) and the other among process countermeasures (to send short status notes per email).

5. Summary and Discussion

This paper presents some of the major results of an online survey among IT professionals who are experienced in distributed software development. In doing so, we focused on the results concerning requirements engineering. The participation rate in this survey was high. So was the average experience of the participants with distributed software development. This shows the practical relevance and representativeness of the topic.

We identified five barriers which influence distributed software engineering projects: process barriers, cultural barriers, domain specific barriers, technical barriers as well as communication barriers. Comparing a former interview study with our study, the authors in [2] also identify communication and domain knowledge issues when developing large software systems (not necessarily in distributed teams). In addition to the problems mentioned in [2], in our study we identified three further problems, which often occur within a distributed project context. The most frequently reported problems concern *process* barriers. Thus, documented processes are not efficient and appropriate in a distributed context with the result that documented processes are not actually implemented. Another issue mentioned by the participants of our study is related to *cultural* barriers. This is not surprising, as the study in [2] did not analyse distributed projects. A project team working at one location is more likely to be homogeneous with respect to cultural characteristics than the members of a geographically distributed project. In contrast to the study in [2], the respondents of our study also report *technical* barriers. Above all, difficulties to provide consistency of distributed data as well as the lack of support for distributed processes are the main issues mentioned by the respondents. In contrast to the study in [2], conflicting requirements are not often mentioned by the respondents. Thus, respondents of our study do not consider this specific to distributed software development.

Altogether, our study shows that problems related to distributed software engineering in general and specific problems related to requirements engineering are similar, but their relative occurrence frequencies vary. For instance, communication barriers are more important in requirements engineering and technical barriers are less important. Moreover, there are particular problems related to requirements engineering. Above all, communication plays a critical role as an important measure against problems.

Our study shows that process-related and human aspects are more important than technical ones. The survey participants did not emphasize tool support, when answering to open questions about problems which they encountered. As communication has been such a frequent countermeasure to many different problems in requirements engineering, we conclude that the main goal of tools and processes must be to support communication.

As can be seen from the literature cited in the introduction, it is not surprising that communication plays such an important role in distributed software development, as a frequent type of barrier as well as a recommended countermeasure to overcome barriers. Thus, this work confirms former, often anecdotic and qualitative findings quantitatively.

Barrier	[2]	[4]*	[5]	[6]*	[9]*	[14]*	[19]	[20]*	[22]*
Comm.	X	X	X		X	X	X	X	X
Domain specific	X		X			X		X	
Cultural				X				X	
Technic al			X						X
Process						X	X	X	X

Tab. 2: Barriers/ problem sources identified by other studies for distributed and non-distributed development (studies which investigate distributed development are marked by *)

Other empirical studies of software development found similar barriers as we did. We summarize their results according to our classification in Table 2. Details of the context of the studies and results are given below:

Curtis et al. [2] found three types of problems in the development of large systems: (1) the thin spread of application domain knowledge, (2) fluctuating and conflicting requirements, (3) and communication bottlenecks and breakdowns. Whether these are more frequent in distributed software engineering than in the "large system development" investigated by Curtis et al. cannot be told, because the authors do not measure their importance quantitatively.

An empirical study of distributed software development found the following types of problems [20]: requirements engineering, lack of standards of the activities in distributed teams, the difficulty to share information and the lack of a well-defined software development process, language barriers and communication, cultural differences, context sharing and trust acquisition among teams. A study of distributed RE [22] found these: communication, planning, management, review process, validation, prototyping, traceability, tool support, knowledge management. [14], in the context of distributed development of embedded systems found: time difference, cultural differences, lack of knowledge of the product. These all are consistent with our findings, although the granularity is not always the same (compare to Figure 3), and the other studies do not quantify the importance of the problems.

One interview study of distributed software development identified some further problems, not found in our survey [14]. These are: openness of communication between partners, problem hiding in customer-supplier relations, unclear assignments, trust, agreeing on intellectual property rights, reliability of the partners´ development schedule, continuation of the collaboration in the future, predicting the most sales-boosting features, quality of the product, becoming too dependent on one partner, competence of new partners, weakening of one's own competence. One can wonder whether such problems are rather mentioned in an interview study than in an online survey because of higher trust and openness towards the interviewer, or whether they were not considered to be specific to distributed software development.

As was mentioned in section 4.3, such problems which are specific to RE, but not to distributed RE, were rarely reported during our survey, such as: understanding the users´ needs, conflicts among different customers, how to prioritize requirements, requirements changes [19], or: package considerations (analysis of COTS products), level of detail of process models, examining current system, user participation, managing uncertainty, CASE RE tools, project management [5]. This is because we asked for problems which are specific for distributed RE, and also shows that the participants focused on these.

Counter-measure	[2]	[4]*	[5]	[9]*	[14]*	[20]*	[22]*
Communi-cation			X	X		X	
Face-to-face comm.		X			X	X	
Training	X		X			X	
Tool	X			X			X
Process			X			X	
Others					X		

Tab. 3: Countermeasures identified by other studies for distributed and non-distributed development (studies which investigate distributed development are marked by *)

Table 3 presents an overview of countermeasures proposed by other studies. The following were proposed by practitioners: to increase application domain knowledge, tools and methods must allow change, appropriate communication media [2]; planning, training, standardization, requirements engineering (face to face if possible), trust and integration [20].

Although the other studies did not measure the importance of these countermeasures quantitatively, it seems that tools and communication media as well as training had a lower weight in our study. We believe that this is because these countermeasures are not successful or not perceived as being so by the team members (we explicitly asked for successful countermeasures to the named problems). For instance, in [14] it was found that most tools do not support collaborative development well enough. Although practitioners did not report tools as a major problem neither in [14] nor in our study, this can serve to explain why tools are rarely seen as solution of problems in distributed software development.

From literature, one can edit lists of countermeasures which are more comprehensive than those found in empirical studies [14]. Many of these countermeasures were not mentioned by the practitioners in our study and in those studies cited above. There can be several explanations for this observation. Either these countermeasures are not known to practitioners, are not applied or are not perceived as being useful. Such countermeasures were: synchronisation of main milestones, clear decision-making practices, decoupling the work across different sites, one project leader, relationship management, architectural practices, frequent deliveries, frequent and incremental integration, up-to-date documentation [14].

Possible threats to the validity of our results include for instance, that a high proportion of the participants of this survey were developers and designers. This does not necessarily mean that this adds a significant bias to the results as most of the projects were small; so also the developers probably had an overview of the project, and the questionnaire always offered the option to answer “I do not know”, e.g. concerning requirements engineering problems. However, our analyses in the beginning of section 4.3 show that requirements engineering problems evidently have also been known to participants other than the requirements engineer. The answers to our questions were subjective, i.e. the participants named the problems which were most memorable to them and the countermeasures which they believed were most efficient. The relevance of these problems and the efficiency of countermeasures have not been shown by statistical analyses of project data, which was not our purpose. Some practices which usually are advised in technical literature, like using a glossary in requirements engineering, rarely appeared in the answers. This does not necessarily mean that they are not used in practice, but this shows that their lack has not been a major problem (either because a glossary is less important or used without saying), and that such a practice was not the most important solution to a problem. As we have discussed before, some measures are necessary, but not sufficient preconditions of good work (like the tools) and therefore are presumably not mentioned in this survey. We did not compare our results quantitatively with such for non-distributed projects so far, mainly because our focus was to investigate the state of the practice. Some of the identified problems also occur in non-distributed projects. However, the comments and examples given by the participants indicate that they quite well understood that we asked for problems which were specific to distributed work. The above must be kept in mind when interpreting the survey results. Nevertheless, we think that our results are a good basis for investigating project problems and practices as perceived by the team members, because of the high number of participants and the amount of data.

Future work will focus on further analyses, especially on software development phases other than the requirements engineering. Furthermore, we expect that the in-depth analysis of correlations will lead to further interesting insights, e.g. whether some problems are more frequent in big projects than in small ones.

Acknowledgements

We would like to thank the numerous participants of the survey for their time, trust and meaningful answers, and Barbara Paech for her constructive suggestions, comments and references. We also appreciate the financial and organizational support of the survey by the MFG, especially by Eike Bieber. We also would like to thank Heise for publishing the survey address in their news ticker, which significantly contributed to the high participation. This work was funded by the research network PRIMIUM.

6. References

[1] C. Cook and N. Churcher, “An Extensible Framework for Collaborative Software Engineering”, *Proceedings of the 10th Asia-Pacific Software Engineering Conference APSEC’03*, IEEE, 2003, pp. 290–301.
[2] B. Curtis, H. Krasner, and N. Iscoe, “A field study of the software design process for large systems”, *Communications of the ACM 31 (11)*, 1988, pp.1268-1287.
[3] D.L. Dean, J.D. Lee, M.O. Pendergast, A.M. Hickey, J.F. Nunamaker, “Enabling the Effective Involvement of Multiple Users: Methods and Tools for Collaborative Software Engineering”, *Journal of Management Information Systems 14 (3)*, 1998, pp. 179–222.
[4] A.H. Dutoit, J. Johnstone, and B. Bruegge, “Knowledge scouts: Reducing communication barriers in a distributed software development project”, *Proceedings of the Eighth Asia-Pacific on Software Engineering Conference APSEC*, IEEE, Dec. 2001, pp. 427-430.
[5] K. El Emam and N.H. Madhavji, “A field study of requirements engineering practices in information systems development”, *Proceedings of the International Symposium on Requirements Engineering*, IEEE, 1995, pp.68-80.
[6] J. Grieb and U. Lindemann: “Design Communication in Industry: A Survey Analysis”, *International Conference on Engineering Design ICED 05*, Melbourne, Aug. 15 – 18, 2005, pp. 586-587.
[7] R.E. Grinter, J.D. Herbsleb, and D. E. Perry, “The geography of coordination: Dealing with distance in R&D work”, *Proceedings of the ACM Conference on Supporting Group Work (GROUP 99)*, Phoenix, AZ, November 14-17, pp. 306-315.
[8] J. Grudin, “Why CSCW applications fail: problems in the design and evaluation of organization of organizational interfaces“, *Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work*, ACM, New York, 1988, pp.85-93.
[9] C. Gutwin, R. Penner, and Kevin Schneider, “Group Awareness in Distributed Software Development”, *Proceedings of the 2004 ACM conference on Computer supported cooperative work CSCW’04*, IEEE, Chicago, Illinois, USA, November 6–10, 2004, pp.72-81.
[10] A. Herrmann, D. Kerkow, and J. Doerr, “Exploring the Characteristics of NFR Methods – a Dialogue about two Approaches”, *REFSQ - Workshop on Requirements Engineering for Software Quality (2007), Foundations of Software Quality*, to be published.

[11] A. Herrmann, B. Paech, “Quality Misuse”, *REFSQ - Workshop on Requirements Engineering for Software Quality (2005), Foundations of Software Quality, Essener Informatik Beiträge,* Universität Duisburg-Essen, Essen, 2005, pp. 193-199.

[12] T. Hildenbrand, F. Rothlauf and A. Heinzl, “Ansätze zur kollaborativen Softwareerstellung”, WIRTSCHAFTS-INFORMATIK 49 (Special Issue), 2007, pp. S72–S80.

[13] I. Hoh and R. Siddharta Roy, “Visualization issues for software requirements negotiation”, *Proceedings of the 25th Annual International Computer Software and Applications Conference COMPSAC,* 8-12 Oct 2001, pp.10-15.

[14] J. Hyysalo, P. Parviainen, and M. Tihinen, “Collaborative Embedded Systems Development: Survey of State of the Practice”, Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS06), IEEE, 2006.

[15] IEEE, *Std. 830-1998: IEEE Recommended Practice for Software Requirements Specification*, IEEE, Washington, 1998

[16] MFG Baden-Württemberg, http://www.english.doit-online.de/cms/About+us/MFG+Baden-W%FCrttemberg, last visited May 2007

[17] B.A. Kitchenham and S.L. Pfleeger "Principles of survey research: part 3: constructing a survey instrument", *SIGSOFT Softw. Eng. Notes* vol. 27, no. 2, pp. 20-24, 2002.

[18] B. Kitchenham and S.L. Pfleeger "Principles of survey research part 6: data analysis", *SIGSOFT Softw. Eng. Notes* vol. 28, no. 2, 2003, pp. 24-27.

[19] M. Lubars, C. Potts, and Ch. Richter, “A review of the state of the practice in requirements modeling”, *Proceedings of the International Symposium on Requirements Engineering,* IEEE, 1992, pp.2-14.

[20] R. Prikladnicki, J.L.N. Audy, and R. Evaristo, “An Empirical Study on Global Software Development: Offshore Insourcing of IT Projects”, *Proceedings of the International Workshop on Global Software Development,* International Conference on Software Engineering (ICSE 2004), IEEE, Edinburgh, Scotland, May 24, 2004, pp. 53-58.

[21] B. Regnell, M. Höst, J. Natt och Dag, P. Beremark, and T. Hjelm, ”An Industrial Case Study on Distributed Prioritisation in Market-Driven Requirements Engineering for Packaged Software, *Requirements Engineering Journal 6(1)*, 2001, pp. 51–62.

[22] D. Zowghi, D. Damian, and R. Offen, „Field Studies of Requirements Engineering in a Multi-Site Software Development Organization“, *Proceedings of the Australian Workshop on Requirements Engineering,* Sydney, Nov. 2001.