2010 18th IEEE International Requirements Engineering Conference

Comparison of Requirements Hand-Off, Analysis, and Negotiation: Case Study

Samuel Fricker and Martin Glinz
Department of Informatics, University of Zurich, CH-8050 Zurich, Switzerland
Email: {fricker, glinz}@ifi.uzh.ch

***Abstract*—Companies in the software business often distribute requirements engineering responsibilities over several roles. Product management has overall product responsibility and performs early-phase market-driven requirements engineering. Product development implements the product and performs late-phase solution-oriented requirements engineering. Such shared responsibility provides advantages in the utilization of specific knowledge, skills, and resources, but leads to problems of mutual understanding and coordination.**

Earlier research proposed a negotiation process, handshaking with implementation proposals, that allows product management and development to achieve agreed requirements understanding. The process found acceptance in industry, but the relative advantages compared with traditional requirements hand-off and analysis had not been understood yet. This paper fills this gap by describing a case of measuring requirements and design volatility and an architect's requirements understanding during requirements hand-off, analysis, and negotiation.

Keywords-requirements specification; requirements communication; requirements negotiation; empirical study

I. INTRODUCTION

Software businesses often distribute requirements engineering responsibilities over several roles [1]. Well established is the collaboration of product management concerned with market needs [2] and product development concerned of the technological aspects of a product [3]. Such shared responsibility provides advantages in the utilization of specific knowledge, skills, and resources. However, it leads to problems of mutual understanding and to coordination needs between the two roles.

For a given product, product management is responsible for early-phase requirements engineering [4]. Goal structuring [5] is employed for continuous elicitation, triage, analysis, and management of requirements that are used for roadmapping and release planning of software products [6]. Product development is responsible for late-phase requirements engineering of given product releases. It uses software specifications techniques [7-9] to prescribe structure, behavior, and desired properties of the solution that is implemented by ensuing development projects.

The interdependencies between product management and development require intertwining early-phase and late-phase requirements engineering [10]. Requirements communication based on requirements hand-off from product management to development risks leading to unsatisfactory results. It may be complemented with system analysis [8] and with negotiation of implementation proposals [11]. Implementation proposals are a form of coarse-grained traceability between early- and late-phase requirements. Handshaking, the combination of these practices to establish a shared requirements understanding, was successfully transferred to industry [12], without understanding the practices' relative advantages however. This leaves questions unanswered regarding the value of such added investment upfront of a development project.

The paper provides method selection support by comparing the relative effect of requirements hand-off, analysis, and negotiation on requirements and design volatility and on a solution architect's requirements understanding. It describes a case in which these effects were measured, facilitated by expressing early-phase requirements with goal models, late-phase requirements with a solution-oriented system specification, and traceability between these requirements with implementation proposals. The case showed that the three practices can be regarded as steps in a win-win negotiation process. Hand-off corresponded to product management positioning, analysis to product development positioning, and negotiation to position alignment. Omission of a practice would have led to delays, system value, or acceptance problems.

Section II introduces traceability between early- and late-phase requirements. Section III describes the planning, section IV the operation, and section V the analysis and discussion of the requirements communication case. Section VI discusses the results. Section VII concludes.

II. EARLY- TO LATE-PHASE REQUIREMENTS TRACEABILITY

A formalization of early- and late-phase requirements and traces is needed to measure the effects of requirements hand-off, analysis, and negotiation on requirements and design volatility and on requirements understanding in a repeatable manner. The here employed approach for documenting requirements and traces is a formalization of the Handshaking method [11, 12]. A product manager is expected in this formalization to expresses early-phase requirements with the requirements abstraction model (RAM) [5], a simple goal modeling paradigm employed in a product requirements management. RAM ensures that requirements are sufficiently atomized and that means and ends are not confounded, hence allowing observing volatility of comparable requirements. Development expresses late-phase requirements with ADORA [8], a system specification language that, in contrast to UML for example, ensures specification consistency. Implementation proposals are constructed by referring to RAM-based requirements and to ADORA model fragments.

1090-705X/10 $26.00 © 2010 IEEE
DOI 10.1109/RE.2010.29

A. Requirements Abstraction Model

The requirements abstraction model (RAM) [5] is a simple goal modeling paradigm that supports continuous product requirements engineering by distinguishing four requirements abstraction levels. It has successfully been used in industry where it yielded all-over-the-board improvements compared with unstructured requirements management [13].

Figure 1 shows an extract of the industrial RAM model developed during the case described in Section IV, a license management system. The requirements on higher abstraction levels motivate requirements on the next-lower abstraction level. The highest abstraction level relates to company strategy, the lowest to product design. This traceability from design to strategy ensures that product design supports the achievement of company objectives.

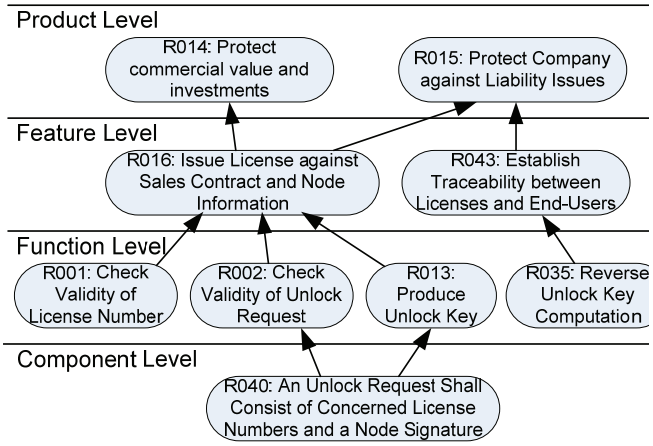

Figure 1. RAM-based early-phase requirements.

RAM allows observing whether a product manager has changed his interests and expectations by monitoring the requirements placed at a given abstraction level. High abstract levels capture interests, low levels expectations towards development.

B. ADORA and Design Decisions

ADORA [8] is a hierarchical approach to object-oriented modeling of a software system that integrates functional, structural and behavioral aspects of the system into one coherent model. Late-phase requirements are specified with ADORA in terms of actors, objects, scenarios, and states. ADORA allows such modeling with flexible degree of detail for incrementally validating completeness, feasibility, and acceptance of the system to be implemented. The decisions captured in such an archetype model for the system to be developed are subsumed by the term *design* in this paper.

Figure 2 shows a high-level ADORA model for the system developed during the case described in section IV. A 'license manager' was planned to be built that consisted of a 'license server', a 'sales server communication module' connected to a set of 'customer'-operated 'external sales servers', a set of 'customer support'-operated 'thin clients', a 'product communication module' that communicated with 'protected products', and a set of 'Enterprise Resource Planning (ERP) interfaces' that communicated with 'ERP systems'.

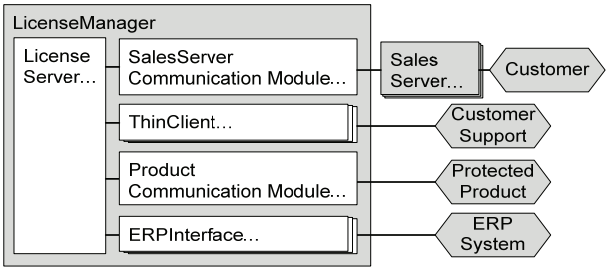

Figure 2. High-level system specification with ADORA.

ADORA allows a development team to document interests in terms of planned product design. The specification further supports analysis of advantages, limitations, risks, and implementation effort of the specified system.

An ADORA model consists of a number of design decisions [14-16] that correspond to requirements placed on low RAM abstraction levels. Taxonomies of design decision types characterize typical system specification concerns [17]. TABLE I summarizes the ADORA support for common design decision types. TABLE II shows how the design from Figure 2 can be expressed with design decisions.

TABLE I. MAPPING OF DESIGN DECISION TYPES TO ADORA CONCEPTS.

Existence	
Functionality	Scenario, State
Parts	Object, Object Set, Element of Environment (Actor)
Data	Signal, Annotation for Model Element
Structure	
Relationships	Nesting, Association, Scenario Relationships, State Transition
Architectural Style	Annotation (Tag) for Set of Model Elements
Design Pattern	Annotation (Tag) for Set of Model Elements
Properties	
Design Rule	Annotation (Tag) for Model Element
Technology Use	Annotation (Tag) for Model Element
Texture	
Interface	Annotation (Tag) for Model Element
Metaphor	Annotation (Tag) for Model Element

TABLE II. DESIGN FROM FIGURE 2 EXPRESSED WITH DESIGN DECISIONS.

Existence	
Parts	- LicenseManager, Sales Server, License Server, SalesServer Communication Module, ThinClients, Product Communication Module, ERPInterfaces; - Customer, Customer Support, Protected Product ERP System.
Structure	
Relationships	- License Server part of License Manager, ThinClient part of LicenseManager, etc.; - License Server connected to ThinClient, ThinClient connected to Customer Support, etc.

ADORA models allow observing whether the development team has changed its intentions. Changed design decisions imply changed development intentions.

C. Implementation Proposals

Implementation proposals (IP) support requirements communication by capturing tentative or decided agreements between product management and development for given negotiation themes [12]. An IP is created for a negotiation theme and captures coarse-grained traceability between relevant requirements and design. It can be enhanced by

documenting negotiation and planning concerns. IP are negotiated one theme after the other until sufficient coverage of requirements is achieved. This ultimately leads to win-win decisions and a trusted customer–supplier relationship.

The following subsections discuss the IPs in terms of their basic structure, negotiation support and other attributes. Figures 3-5 show examples that are simplified versions of the IPs created in the study described in section IV.

1) Basic Structure: IP document how given *design* proposed by a development team contributes to given *requirements* that are provided by a product manager. Design decisions and requirements stand in a many-to-many relationship to each other. One design decision can be motivated by a number of requirements. One requirement can be implemented by combinations of design decisions.

IP balance richness and flexibility for handling variants by grouping requirement–design traces into *themes*. A theme can be dedicated to an important part of the solution, to a product feature, to a development increment, or to any other concern that is meaningful to the communicating parties. The theme is documented by the IP's title.

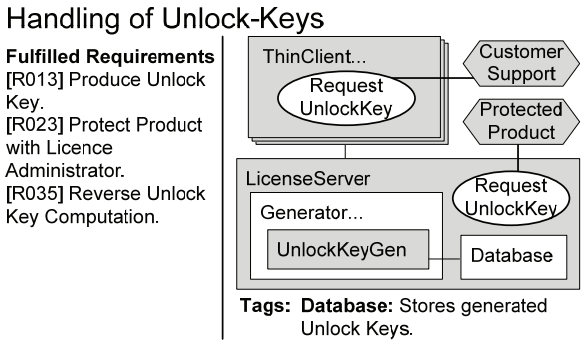

Figure 3. *Handling of UnlockKeys* implementation proposal (simplified).

Figure 3 shows an IP for the theme 'handling of unlock keys'. It describes functionality, parts, and data that an architect has proposed and relates that design to justifying requirements provided by a product manager. The design concerned the 'thin clients' introduced in Figure 2 that interacted with the stakeholder 'customer support' within the scenario 'request unlock key'. Connected to the 'thin clients' was the 'license server' introduced in Figure 2 that contained a 'generator' that then contained an 'unlock key generator' connected to a 'database'. The 'license server' interacted with a 'protected product' within a second scenario 'request unlock key'. The design fulfilled the requirements R013 'product unlock key', R023 'protect product with license administrator', and R035 'reverse unlock key computation'.

2) Negotiation Support: The first interpretation of handed-off requirements is often not perfect. A proposed solution may be based on wrong assumptions and have unacceptable side effects.

The fewer requirements are handed-off to development, the more assumptions need to be taken for justifying proposed design. These assumptions state the conditions under which the design is meaningful. Confirmed assumptions may be turned into requirements.

The thinner the product manager's understanding of solution technologies and architecture is, the more difficult it is for her to predict their impact on product utilization and context. Solution concepts can provide surprising advantages, disadvantages, and risks, hence need to be discussed and agreed during the requirement negotiations. Accepted impact may be turned into requirements.

Figure 4 shows an IP for the theme 'thin client' justified with assumptions and impact considerations. The proposed design concerned the previously mentioned 'thin clients' that were connected to the 'license server' and interacted with 'customer support' within the scenario 'request unlock key'.

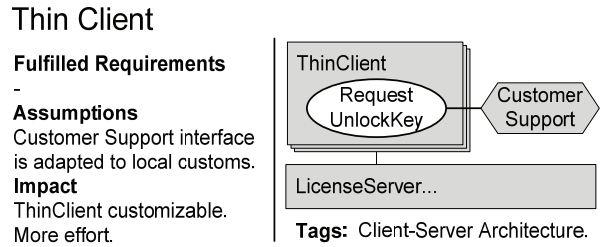

Figure 4. Implementation proposal based on assumptions and impact.

An IP describes a preferred design and lists dismissed designs with the reasons for their dismissal. Figure 5 shows the IP 'thin client' that resulted from development discussing the IP shown in Figure 4 with product management. The preferred alternative was justified with the now discovered requirement R059 'customizable interface' that replaced the assumptions and impact from Figure 4.

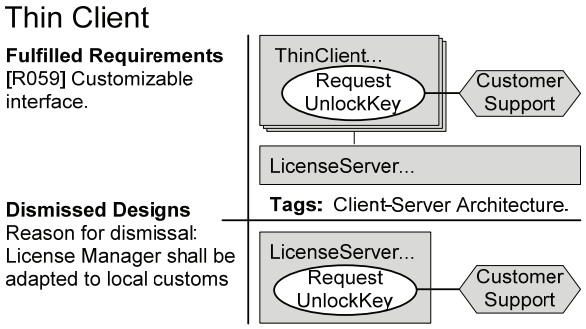

Figure 5. Implementation proposal with preferred and dismissed designs.

3) Other Attributes: Not only traceability, assumptions, design impact, and alternatives need to be managed in requirements communication, but also the communication process itself, effort estimation, project planning, acceptance testing, and integration of requirements communication into other company-specific processes. IP support these concerns with additional specific attributes [12].

IPs allow observing how requirements are aligned with design. The coarse-grained traces indicate how far design covers requirements and how far requirements cover design. The more requirements are covered by agreed IPs the more completely requirements understanding has been established. The history of changes to the traces shows how requirements understanding has changed.

III. REQUIREMENTS COMMUNCATION CASE: PLANNING

Adoption of good requirements communication practices requires an understanding of the practices' relative effects on a solution architect's requirements understanding. Good understanding of requirements requires in addition that requirements and design volatility have been stabilized. Case study research [18] has been employed to explore the relative effects of requirements hand-off, analysis, and negotiation on these factors. This form of research provides rich insights into how and why the practices have influenced requirements understanding. The ultimate objective was to provide method selection advice to practitioners confronted with requirements communication.

The study answers the following research questions. Q1a: *How much does the presentation of requirements influence design?* Q1b: *How much does the presentation of design influence requirements*? Q2: *Are requirement–design traces useful for measuring how much requirements are understood by the requirements receiver?* A lack of influence of requirements on design (low Q1a) would imply that a customer cannot push requirements to a supplier using requirements hand-off. Unidirectional influence of requirements on design (low Q1b and high Q1a) would imply that hand-off based on unambiguous requirements specification [19] followed by requirements and system analysis suffices for successful requirements communication. Reciprocal influence of requirements and design (high Q1b and high Q1a) would imply that requirements negotiation is critical for requirements communication. Affirmation of Q2 would encourage the use of traceability-based measurements, rather than measurements of the requirements specification [7], for managing requirements communication.

Q1 and Q2 are answered by studying the effect of the treatments in the sequence described in TABLE III. This sequence exposes the relative effect of requirements and system analysis (PO2) to requirements hand-off (HO) and of requirements negotiation (NE) to requirements and system analysis (PO2). The sequence allows answering Q1 because definition of requirements and design and their exposure to the collaboration partner can be controlled. Q2 is answered by comparing the evolving degree of agreed traceability between requirements and design with the architect's subjective perception of requirements understanding.

TABLE III. CASE STUDY PHASES AND TREATMENTS.

Phase	Treatment
1. Positioning 1 (PO1)	Product manager (PM) prepares requirements
2. Hand-off (HO)	PM hands requirements off to architect
3. Positioning 2 (PO2)	Architect analyzes requirements and system
4. Negotiation (NE)	Both negotiate requirements and design
5. Confirmation (CO)	Both document negotiation results

The following data was collected along the process described in TABLE III. Requirements and ADORA model baselines were used to study the effects of the treatments on requirements and design, hence to answer Q1b and Q1a. Implementation proposal baselines and architect interviews were used to understand the degree of the architect's requirements understanding, hence to answer Q2. Both parties were interviewed after the study to understand their stance towards the experienced case.

TABLE IV describes the measurements applied on the collected data for answering the research questions. The measurements are based on the concepts presented in section II. Requirements volatility is measured on the RAM feature level. Design volatility is measured in terms of ADORA model elements and text annotations of these elements. Traceability is measured by analyzing the implementation proposals.

TABLE IV. MEASUREMENTS.

Measurement	Scale	Levels
Q1a: Degree of Influence of Requirements Presentation on Design		
Design volatility in given phase	Fraction of changed design decisions	0%: treatment without effect 100%: treatment dominant
Q1b: Degree of Influence of Design Presentation on Requirements		
Requirements volatility in given phase	Fraction of changed requirements	0%: treatment without effect 100%: treatment dominant
Q2: Correlation of Traceability with Perceived Requirements Understanding		
Architect's perceived ability to develop accepted solution (Exec-Ability)	Rating 0% - 100%	0%: no knowledge 60%: able to develop solution 100%: perfect understanding
Coverage of requirements by design (R-Cov)	Fraction of requirements used to justify design	0%: no coverage 100%: complete coverage

The study was performed with early-phase requirements prepared for an industrial software development project. A product manager and a solution architect followed the process described in TABLE III. The product manager was a practitioner with more than 20 years experience in a global Fortune500 company in the power technologies domain. He participated for improving his requirements communication skills. He intended to develop the specified product. The company used the requirements abstraction model to structure product requirements and had a tradition in modeling systems with graphical languages. The solution architect was a student with ½ year software engineering experience. He participated for learning about requirements engineering and for receiving study credits. His grading depended partially on the quality of solution design.

Participant selection followed the intensity and critical case strategies. It ensures minimal common background and full control of communication. It represents an extreme case of imbalance of seniority. If Q1b is high for the chosen solution architect, it is likely to be high for situations where product management can less dominate.

The study started with a briefing about research objectives and the requirements communication process. The specific research questions Q1a, Q1b, and Q2 were not communicated, however. The study was finalized by debriefing the participants, analyzing collected data, and validating the results with the study participants.

A. Threats to Validity

1) Construct Validity: The study design balances different validity threats and gives measurement of requirements and design evolution first priority. High requirements atomicity and proper separation of abstraction levels are facilitated by employing RAM. Design changes are followed by monitoring changes in the ADORA models.

The use of ADORA further mitigates problems like consistency of the system specification [8].

The study design reduces the likelihood that attempts of persuading the product manager of a given solution conceal lack of requirements understanding. An inexperienced architect can't easily manipulate an experienced product manager. Hence, the study results that concern the effects of design on requirements are defensive and likely to be amplified with increased architect experience.

2) Conclusion Validity: Control of the process outlined in TABLE III and of the communication between product manager and architect is facilitated by working with a student architect. This control eliminates confounding effects that emerge from a shared background and contact network between the parties and from communication other than HO and NE. It also facilitates proper use of ADORA and implementation proposals, contributing to data reliability.

Exec-Ability is a measure typically applied in industrial practice for judging requirements understanding, but carries the risk that measurements cannot be repeated with exactly the same results. Interviews regarding the meaning of the obtained values for the architect and the rationale for the values given were used to increase measure reliability.

3) Internal Validity: A rich picture of the evolving requirements understanding is provided by complementing quantitative data with qualitative data about the behavior and reflections of the product manager and the architect. The qualitative data allows knowing the tactics employed for achieving agreed requirements understanding, for understanding the reasons for the study outcomes, and for posing new questions that become apparent from the increased understanding of requirements communication

4) External Validity: Measurability and control of the study operations are traded off to some degree against realism with respect to industrial practice. An inexperienced architect cannot contribute deep product knowledge. Use of the implementation proposal's negotiation attributes *assumptions* and *impact* requires such domain knowledge. Hence, the study results that concern these attributes are defensive and likely to be amplified in a full industrial context, for example with increased architect experience. Use of the basic *requirements*, *design*, and *theme* attributes is not affected.

The study focuses on the technical content of requirements and design and excludes budget and effort negotiations. Scarce budget encourages deletion of low-value requirements and corresponding implementation. Hence, the study results that concern such requirements and design changes are defensive and likely to be amplified.

Case studies generalize through models or theories that are explored with rich data, and not statistically through a high number of data points. The here presented case study provides insights into evolving requirements, design, and mutual understanding by making the process and changes transparent, describing examples, and by reporting opinions of the participants.

IV. REQUIREMENTS COMMUNCATION CASE: OPERATION

A. Positioning 1 (PO1)

The product manager prepared 51 requirements for a license management system in a spreadsheet according to RAM principles. Figure 1 shows some of these requirements. The product manager collected, specified, and agreed the requirements with stakeholders during ½ calendar-year with approximately 1 person-month work effort. An experienced requirements engineer ensured that the requirements specification was of high quality.

Column R-Post-PO1 in TABLE V lists the distribution of the 51 requirements over the RAM abstraction levels. The requirements below the product level were linked with contribution links to requirements on the next-higher abstraction level. A glossary with 21 definitions and a business process specification that described the interplay of 5 of the function-level requirements complemented the requirements specification.

TABLE V. EVOLVING NUMBER OF REQUIREMENTS (*: REQUIREMENTS AFFECTED BY LOWER-LEVEL REQUIREMENT CHANGES IN PHASE NE).

Abstraction Level	All Requirements		Requirements Addressed by IPs	
	R-Post-PO1	R-Post-NE	IP-Post-PO2	IP-Post-NE
Product-Level: Business goals	8	8	*6	*7
Feature-Level: Product goals	16	30	5+*1	22+*2
Function-Level: Usage of solution	23	41	4	30
Component-Level: Ideas for solution design	4	7	0	4
Total	51	86	11	65

B. Hand-Off (HO)

The product manager handed the requirements over to the architect. The product manager explained in a one-hour meeting the purpose of the product, discussed the business process and walked through all requirements. The architect asked two questions. The meeting had an open and friendly atmosphere, which was enabled by preceding informal coffee-corner discussions.

The architect felt after the meeting that he started to grasp the expectations towards the system, but that his level of requirements understanding, in terms of his ability to develop an acceptable product, was still very low (row HO in TABLE VI). He felt that the meeting was too short to absorb the amount of presented information. He was not able to build a consistent mental picture of the system yet.

TABLE VI. ARCHITECT'S EFFORT AND PERCEIVED ABILITY TO EXECUTE.

Phase	Invested Effort (hours)	Perceived Ability to Execute (Exec-Ability)
PO1	0h	0%
HO	1h	20%
PO2	64h	55%
NE	2h	65%
CO	4h	70%

C. Positioning 2 (PO2)

The architect specified the system intended to fulfill the requirements by incrementally designing the solution with the ADORA language and tool and by separately documenting non-ADORA supported design decisions. He used object-oriented design principles and regularly reviewed the solution against requirements. The creators of ADORA ensured mastery of the language. The resulting model expressed the architect's opinion of what should be built. Figure 2 shows a high-level view of the model. Column Post-PO2 (prepared) of TABLE VII lists the frequency of the design decisions types he employed.

TABLE VII. EVOLUTION OF SYSTEM SPECIFICATION (*: NON-ADORA).

Type of Design Decision	System Specification			Implementation proposals (IP)			
	Post-PO2 prepared	Post-PO2 negotiated	Post-NE negotiated	IP1	IP2	IP3	IP4
Total number	447	126	161	33	34	47	66
Functionality Decisions							
Scenarios	13	13	15	3	4	4	4
Scenario Relationships	2	2	2	0	2	0	0
States	82	0	0	0	0	0	0
State Transitions	114	0	0	0	0	0	0
Structure Decisions							
Objects	20	16	17	6	6	10	13
Object Sets	1	1	3	1	1	0	0
Nestings	117	29	34	9	10	13	16
Associations	23	11	21	7	6	3	3
Actors	9	6	7	4	3	1	1
Architectural Style*	1	1	1	1	1	0	0
Data Decisions							
Signals	18	0	0	0	0	0	0
Entities and Attributes*	34	34	34	0	0	11	23
Data Flow*	10	10	20	0	0	5	5
Property Decisions							
Design Rules*	3	3	7	2	1	0	1

The architect prepared the ensuing negotiation phase with four implementation proposals (columns IP1-IP4 in TABLE VII) by following the structure described in section II.C. The implementation proposal *IP1* described the overview and *IP2*, *IP3*, and *IP4* one major part of the system. Figure 3 gives a representative example of such an implementation proposal, which has been simplified for presentation and confidentiality reasons. The implementation proposals established traceability between requirements and design and contained those design decisions that the architect believed to be relevant for negotiating the solution with the product manager. Column Post-PO2 (negotiated) in TABLE VII shows how much of the overall system specification was covered by the four implementation proposals. Column IP-Post-PO2 in TABLE V shows how many of the requirements were covered by the four implementation proposals.

PO2 lasted 17 weekdays and required 64 person-hours effort (47% workload). The architect believed that he had increased, but not reached a satisfactory level of requirements understanding yet (row PO2 in TABLE VI). He perceived the situation uncomfortable that many design decisions were based on assumptions.

D. Negotiation (NE)

NE was performed in one 2-hour meeting between the product manager and the architect. Again the meeting had an open and friendly atmosphere. They reviewed the four prepared implementation proposals. Figure 6 shows how rich media and face-to-face contact were used for the requirements and design negotiations. Implementation proposal-describing slides, requirements printouts, and ad-hoc drawings on paper were used to point to, criticize, and modify design decisions and requirements. Physical proximity was used to enhance spoken statements with body language that expressed importance and certainty.

Figure 6. Negotiation meeting with rich media.

TABLE VIII. CHANGES FROM PRE- TO POST-NEGOTIATION POSITIONS.

Type of Change	Number of Require-ments (#R)	Negotiated Fraction of Require-ments (%R)	Number of Design Decisions (#DD)	Negotiated Fraction of ADORA Model (%DD)
Stable	51	100%	119	94%
Modified	0	0%	7	6%
Deleted	0	0%	0	0%
Added	35	68%	35	28%
Additionally Covered by IPs	14	27%	0	0%

The discussions led to modifications and additions of the proposed design (columns #DD and %DD in TABLE VIII and column Post-NE negotiated in TABLE VII). Previously unstated requirements (column R-Post-NE in TABLE V) and lack of requirements understanding motivated these changes. Clarifications also helped to link non-understood requirements to the design (last row in columns #R and %R in TABLE VIII). Column IP-Post-NE in TABLE V shows the resulting total number of requirements addressed by implementation proposals.

The architect perceived his requirements understanding after the negotiation meeting just good enough to run a project for implementing an acceptable solution (row NE in TABLE VI). He got so confident because the product manager accepted the majority of the proposed design.

E. Confirmation (CO)

The architect revised the implementation proposals and documented the previously agreed requirements and design changes.

CO lasted 5 calendar days and required ½ person-day effort (17% workload). The revision of the implementation proposals further increased the architect's confidence in his requirements understanding (row CO in TABLE VI).

F. Finalization

The product manager confirmed the study results. He was satisfied with the experience. He continued to develop the requirements by negotiating with two experienced teams with license management technologies. Further adjustments of the requirements reduced development cost and better utilized the capabilities of the teams and of their solutions.

Also the architect confirmed the study results. He stated that experience in the license management domain would have enabled him to be more active during HO. The implementation proposals helped him, however, to become a useful discussion partner in NE. He valued the confirmation he received from the product manager that the proposed system satisfied the product manager's intentions.

V. Requirements Communication Case: Analysis and Interpretation

The presented case provides insights in the collaboration of the product manager in the customer role and the architect in the supplier role during requirements communication. It describes the parties' effort profiles, the reciprocal influence of requirements and design knowledge (Q1a and Q1b), and the effect of hand-off (HO), analysis (PO2), and negotiation (NE) on the architect's requirements understanding (Q2).

A. Effort and Collaboration Profile

Requirements communication was structured as a negotiation process and attempted to minimize the interaction between the negotiating parties. Figure 7 shows the effort profile of the product manager and Figure 8 of the architect. Less effort was devoted to NE than the 80% suggested by negotiation literature [20], a result from effective preparation or a consequence of the study design.

The product manager invested 96% of his effort and the architect 90% to define their negotiation positions. The product manager generated ideas, specified requirements, and negotiated with stakeholders during PO1. The architect, learned ADORA, consulted domain literature, and specified the proposed system in PO2. The positioning was performed independently and was necessary for successful negotiation.

Requirements communication necessitated only 3 hours of direct interaction (HO and NE) and corresponded to roughly 4% of the effort for each of the two roles. HO established contact between the product manager and the architect and launched PO2. The architect got to know the product manager's negotiation style and a feeling for how to communicate with him. The trust-enabling activities of having a coffee together and sharing personal background preceded the meeting. NE allowed sharing and aligning requirements and design knowledge. The face-to-face situation and the use of rich media permitted ad-hoc documentation and pointing to concepts under discussion. This increased the precision of the discussions and ultimately negotiation efficiency and the exploration of alternatives.

Increased interaction between the two parties is likely to shorten the requirements communication effort. Early feedback during PO1 would have allowed the product manager to focus and tailor the requirements specification for the specific architect. Architect experience and early feedback from the product manager would have shortened PO2. More interaction, however, necessitates geographic proximity and great availability for meetings, factors which often are only given in small-scale collocated development.

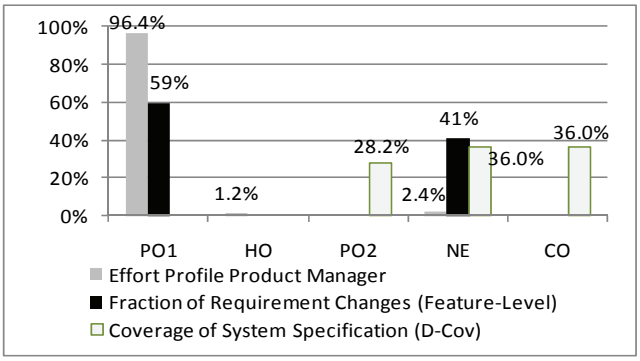

Figure 7. Effect of treatments on product manager's concerns.

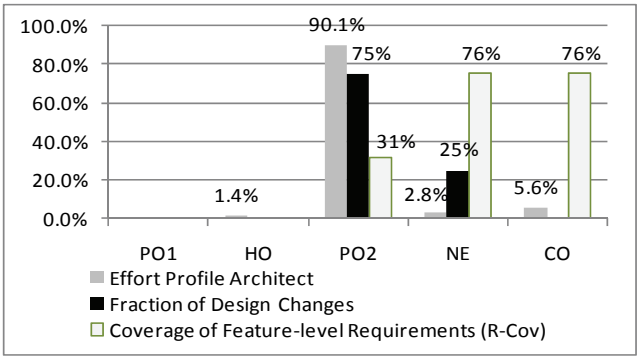

Figure 8. Effect of treatments on architect's concerns.

B. Effect of Design Presentation on Requirements

Design knowledge triggered requirement changes. The 51 requirements produced during PO1 accounted for only 59% of all requirement changes. The 35 requirements added as a result of NE accounted for 41%. Effort negotiations would have introduced even more requirement changes due to deletion of low-priority requirements. Figure 7 shows that the 41% change was achieved with just 2.4% of effort.

Only 36% of the system specification was justified by the requirements before NE. States, state transitions, nesting of states, and signals were not used to specify the implementation proposals. These design decision types made up for almost all disregarded design. Hence, some decision types were perceived to be meaningful for negotiation and others redundant or not useful.

The many requirements changes during NE show that the product manager understood design by discussing implementation proposals. Such requirements change was not possible during HO because he did not understand how the requirements impacted the solution.

Knowledge of the intended solution triggered requirements elaboration and changed the product manager's position. Added requirements provided rationale for why the initially proposed solution contradicted with his intentions and motivated proposed design modifications. The product manager's post-study activities show that he tried to utilize knowledge and capabilities of development teams to improve requirements and consequently the product. He then did not only add requirements to justify changes to the solution, but also modified and deleted requirements to exploit strengths and to account for weaknesses of a possible solution.

C. *Effect of Requirements Presentation on Design*

Changes in requirements understanding caused design changes. The system design defined during PO2 made use of 31% of the feature-level requirements and accounted for 75% of all design changes (Figure 8). NE led to 76% requirements coverage and to 25% design changes. More known requirements were traced and new ones identified.

Even-though the architect was able to produce a large fraction of acceptable design, he only understood a minority of the requirements. The not understood requirements affected a minor, but significant part of the design. NE significantly increased the share of understood requirements, led to previously unstated requirements, and allowed to correct design. The added requirements and rapid feedback on design proposals triggered the necessary design changes.

The many design changes during NE also show that the architect had acquired sufficient knowledge in the license management domain to be receptive for additional input, hence to change his position. This was not possible during HO when he had no understanding of the domain, hence could not absorb the large amount of presented information.

The design did not explicitly trace 24% of the requirements after NE. The traces could have been implicit, the requirements candidates for being deleted, or design changes still pending. Requirements communication may not have been concluded, but would have continued in later project phases to provide the assistance needed to correct errors as early as possible. The effect of this uncertainty on a development project needs to be further investigated.

D. *Towards Measuring Understanding*

Two variables measured evolving requirements understanding: perceived ability to develop an acceptable product (Exec-Ability) and percentage of known feature-level requirements traced by proposed design (R-Cov). Exec-Ability is the architect's subjective confidence for developing an accepted product. R-Cov is opinion-independent and derived from comparing implementation proposals with product requirements. The measures correlate but react differently to PO1, HO, PO2, NE, and CO. Figure 9 shows how the variables evolve.

PO1: Both variables indicated 0% before HO. The architect did neither know the product domain nor the product manager's intentions. No design existed.

HO: Exec-Ability reacted to HO. R-Cov did not change. The architect's perceived understanding increased because he had received the requirements. Still no design existed.

PO2: Exec-Ability's slope was similar to the slope of R-Cov during PO2. Exec-Ability was more optimistic than R-Cov, however. The architect judged at the end of PO2 that he understood the requirements and design almost well enough to successfully run a development project. At the same time product design explicitly considered only 31% of the known feature-level requirements. The architect's perceived understanding was higher than documentation suggested.

NE: R-Cov was very sensitive to NE. Exec-Ability made a small, but important change. 76% of the requirements, including the new discovered ones, justified the design after NE. The architect was confident to be able to develop an accepted but not perfect product. A now started development project would have been likely to be satisfactory.

CO: Exec-Ability was sensitive to CO. The architect's confidence increased again slightly, motivated by a feeling of being confirmed in the thinking he had at the end of NE. R-Cov did not change. No traces were added.

R-Cov appears to be a useful opinion-independent measure of requirements understanding. R-Cov was more pessimistic than Exec-Ability before requirements were aligned with design as a result of NE and more optimistic after NE. The change from pessimistic to optimistic correlated with the small but important change of perceived requirements understanding from just not good-enough to slightly more than good-enough. R-Cov is easily computed using project data without requiring human judgment. The indicator should be further validated, however, by studying its relationship with important success measurements such as software acceptance and project planning accuracy [12].

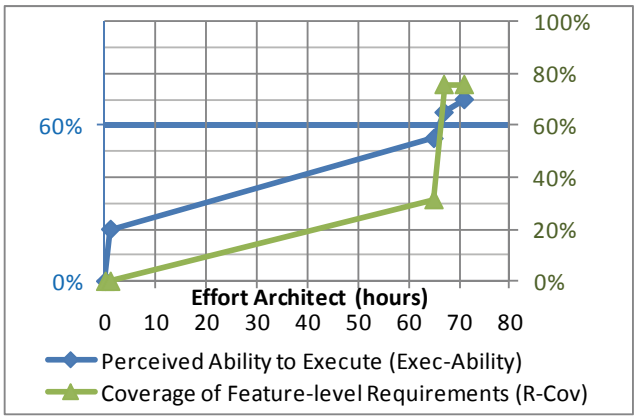

Figure 9. Exec-Ability (60%: good-enough) and R-Cov evolution.

VI. DISCUSSION

A. *Contributions*

The study compared the relative effect of requirements hand-off, analysis, and negotiation between an experienced product manager and an inexperienced architect. Requirements hand-off followed the product manager's positioning and launched analysis. It enabled independent analysis work, but led to clearly insufficient requirements understanding by the architect. Systems analysis supported the architect's positioning. It led to almost good-enough requirements understanding, with only 31% of the

requirements used for justifying the solution, however. Negotiation of implementation proposals aligned the two positions so that the right design fulfilled the right requirements. The presentation of design triggered a large number of requirements changes. Some of these changes were employed to motivate significant changes to the proposed solution. Requirements understanding was perceived good-enough only after this negotiation.

The study further explored the evolution of the indicator R-Cov, the fraction of requirements traced by design, for measuring evolving requirements understanding. R-Cov was pessimistic compared with the architect's subjective ability to successfully implement a solution before requirements negotiation and optimistic afterwards. R-Cov values were very sensitive to requirements negotiation, where requirements and design were aligned, and correlated with the small, but important change in the architect's subjective ability to implement the system from just not good-enough to slightly more than good-enough. This encourages investigating whether calibrated R-Cov threshold values may be used to assure requirements communication quality.

B. Related Work

1) Traceability: Traceability reduces the discontinuity of information between requirements and system specifications that hampers the ability of stakeholders to communicate about a system. The effort for specifying and maintaining traces between single requirements and design decisions [21, 22] is often prohibitive [23], however. Fine-grained traceability links would have been too narrow in the described case for requirements analysis and exploring design alternatives. Traceability between just specification documents, then again, would have been too wide and would have discouraged exploration of alternatives.

Implementation proposals apply the idea of model connectors for transforming requirements to architecture [24] to requirements communication. The here presented study has shown that implementation proposals, a coarse-grained form of traceability, can be employed to document agreement between a product manager and an architect for given negotiation themes. The themes referred to parts of the solution and encouraged the product manager and architect to express and adjust their positions in terms of adjusted design and requirements until an agreement was found.

2) Co-Evolution of Requirements and Design: Delivery of accepted software requires intertwining requirements and design [10]. Goal-oriented reasoning can provide constructive guidance for architects in their design tasks [25]. Rules can be used to ensure consistency of formal specification across abstraction levels [26]. There here presented work considers the alignment of requirements and design not as a formal-analytical, but as a collaborative activity for scalability purposes. It has shown that alignment can be achieved with a negotiation process that involves proposal, change and agreement on traces between requirements and design. The negotiation led to shared requirements understanding, utilized the knowledge of the involved parties, and forced early requirements changes.

Empirical research showed that requirements identification is intimately related to solution generation and detailing [27]. The most important problems of designers related to understanding significance and relationships of requirements and to remembering requirements for use during design. The significant increase of R-Cov, the degree of requirements utilization for justifying solution design, in the here presented study has shown that seeking feedback on design proposals from a stakeholder helps mitigating these problems. The negotiations, further, enabled the product manager to discover previously unstated, but relevant requirements that affected design, adding empirical evidence to earlier work that stated that requirements negotiations can be employed to surface tacit knowledge [28].

3) Quality Assurance: A Only measured quality can be managed. Requirements specification completeness has been proposed earlier to measure requirements understanding [29], but has been challenged because it is subjective, hard to measure, and easily forgeable [30]. The here presented study has evaluated R-Cov, a measurement originally defined for implementation control [31], for measuring requirements understanding. R-Cov appears to be more trustworthy than specification completeness because it can be measured in a repeatable manner and can not be manipulated by the customer or supplier alone if R-Cov is based on agreed implementation proposals. In addition, the small transition of perceived ability to execute from less than to slightly more than sufficient correlated with a sharp increase of the R-Cov value from 31% to 76%. A threshold calibrated with historical values can indicate good-enough requirements understanding, hence save costly surprises down the development road.

C. Limitations

The study was performed in a semi-industrial environment. This setup provided advantages in the control and measurement of the situation, but may create questions in the transferability of the results to industrial practice [32]. Key differences between the chosen setup and industry are the architect's experience, the enforced isolation of the communicating parties, the enforced use of ADORA, and attention on proper measurement of R-Cov.

Greater architect experience encourages persuading the product manager of a solution, hence is likely to amplify the already strong effect of design on requirements. Budget and implementation effort negotiations lead to similar amplification. Unwillingness to cooperate reduces the effect.

Intensive collaboration between product managers and architects increases the number of opportunities for knowledge and information exchange, hence is likely to reduce the communication effort observed in the study.

Design is often formulated pragmatically, and not with a single predetermined specification language. Hence, implementation proposals will vary more and better expose issues to be negotiated, hence will even more effectively lead to mutual understanding than indicated by the study. Such variability, however, will not increase general-purpose understandability of the specifications.

The study did not consider the effect of the reached requirements understanding on the development project. Earlier research indicates, however, that similar R-Cov values may lead to predictable development projects and accepted solutions [12]. The study also did not consider the product manager's understanding of product requirements, which clearly evolved through the exposure of the design, hence limiting the understanding of *why* presentation of design influences requirements.

VII. CONCLUSIONS

The paper has evaluated the relative effect of requirements hand-off, analysis, and negotiation on requirements and design volatility and on requirements understanding. Confrontation of the architect with requirements led to design changes, confrontation of the product manager with solution design to requirement changes. Negotiation of implementation proposals aligned the two positions so that the right design fulfilled the right requirements. The study gave insights into the effort profile of the communication partners and proposed and evaluated an indicator, R-Cov, to measure evolving requirements understanding and to manage requirements communication quality. The results support selection of requirements communication practices to ease challenges in the collaboration between marketing and development [1].

Future research should generalize the discovered knowledge by evaluating the proposed measurements in a substantial number of possibly more complex settings and corroborate R-Cov as a requirements communication quality indicator. Future research should also address the understanding of requirements communication by evaluating the product manager's evolving requirements understanding.

REFERENCES

[1] A. Griffin, and J. Hauser, "Integrating R&D and Marketing: A Review and Analysis of the Literature," *Journal of Product Innovation Management,* vol. 13, pp. 191-215, 1996.

[2] C. Ebert, "Software Product Management," *Crosstalk,* vol. 22, no. 1, pp. 15-19, 2009.

[3] I. Sommerville, *Software Engineering*, 8th ed.: Addison Wesley, 2006.

[4] B. Regnell, P. Beremark, and O. Eklundh, "A Market-Driven Requirements Engineering Process: Results from an Industrial Process Improvement Programme," *Requirements Engineering,* vol. 3, pp. 121-129, 1998.

[5] T. Gorschek, and C. Wohlin, "Requirements Abstraction Model," *Requirements Engineering,* vol. 11, no. 1, pp. 79-101, 2006.

[6] I. van de Weerd, S. Brinkkemper, R. Nieuwenhuis, J. Versendaal, and L. Bijlsma, "Towards a Reference Framework for Software Product Management," in 14th IEEE International Requirements Engineering Conference (RE'06), Minneapolis, 2006.

[7] IEEE 830-1998, *IEEE Recommended Practice for Software Requirements Specifications*, 1998.

[8] M. Glinz, S. Berner, and S. Joos, "Object-Oriented Modeling with ADORA," *Information Systems,* vol. 27, pp. pp.425-444, 2002.

[9] OMG, *Unified Modeling Language (UML) Version 2.1.2*, Object Management Group, 2007.

[10] B. Nuseibeh, "Weaving Together Requirements and Architectures," *Computer,* vol. 34, no. 3, pp. pp.115-117, 2001.

[11] S. Fricker, T. Gorschek, and P. Myllyperkiö, "Handshaking between Software Projects and Stakeholders Using Implementation Proposals," in Intl. Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ'07), Trondheim, Norway, 2007.

[12] S. Fricker, T. Gorschek, C. Byman, and A. Schmidle, "Handshaking with Implementation Proposals: Negotiating Requirements Understanding," *IEEE Software,* vol. 27, no. 2, pp. 72-80, 2010.

[13] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin, "Industry Evaluation of the Requirements Abstraction Model," *Requirements Engineering,* vol. 12, pp. 163-190, 2007.

[14] J. Tyree, and A. Akerman, "Architecture Decisions: Demystifying Architecture," *IEEE Software,* no. March/April 2005, pp. pp.19-26, 2005.

[15] P. Kruchten, P. Lago, and H. van Vliet, "Building Up and Reasoning About Architectural Knowledge," in 2nd Intl. Conference on the Quality of Software Architectures (QoSA 2006), Västerås, Sweden, 2006.

[16] J. S. van der Ven, A. Jansen, J. Nijhuis, and J. Bosch, "Design Decisions: the Bridge between Rationale and Architecture," *Rationale Management in Software Engineering*, A. Dutoit, R. McCall, I. Mistrík, B. Paech, eds.: Springer, 2006.

[17] P. Kruchten, "An Ontology of Architectural Design Decisions in Software Intensive Systems," in 2nd Groningen Workshop on Software Variability, 2004.

[18] R. Yin, *Case Study Research: Design and Methods*, 3rd ed.: SAGE Publications, 2003.

[19] C. Rupp, "Requirements and Psychology," *IEEE Software,* vol. 19, no. 3, pp. 16-18, 2000.

[20] L. Thompson, *The Mind and Heart of the Negotiator*, 3rd ed.: Prentice Hall, 2004.

[21] J. Dick, "Design Traceability," *IEEE Software,* vol. 22, no. 6, pp. 14-16, 2005.

[22] A. Dardenne, S. Fickas, and A. van Lamsweerde, "Goal-Directed Requirements Acquisition," *Science of Computer Programming,* vol. 20, no. 1, pp. pp.3-50, 1991.

[23] O. Gotel, and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," in 1st International Conference on Requirements Engineering (ICRE'94), 1994.

[24] N. Medvidovic, P. Grünbacher, A. Egyed, and B. Boehm, "Bridging Models Across the Software Lifecycle," *The Journal of Systems and Software,* vol. 68, no. 3, pp. 199-215, 2003.

[25] A. Van Lamsweerde, "From System Goals to Software Architecture," *Formal Methods for Software Architectures*, M. Bernardo and P. Inverardi, eds.: Springer, 2003.

[26] E. Sikora, M. Daun, and K. Pohl, "Supporting the Consistent Specification of Scenarios across Multiple Abstraction Levels," in 16th International Working Conference on Requirements Engineering: Foundation for Software Quality (RefsQ 2010), Essen, Germany, 2010.

[27] A. Chakrabarti, S. Morgenstern, and H. Knaab, "Identification and Application of Requirements and Their Impact on the Design Process: A Protocol Study," *Research in Engineering Design,* vol. 15, no. 1, pp. 22-39, 2004.

[28] P. Grünbacher, and R. Briggs, "Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin," in 34th Hawaii International Conference on System Sciences, 2001.

[29] K. Pohl, "The Three Dimensions of Requirements Engineering," *Information Systems,* vol. 19, no. 3, pp. 243-258, 1994.

[30] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos, "Identifying and Measuring Quality in a Software Requirements Specification," in 1st International Software Metrics Symposium, 1993.

[31] R. Costello, and D.-B. Liu, "Metrics for Requirements Engineering," *Journal of Systems and Software,* vol. 29, pp. 39-63, 1995.

[32] M. Höst, B. Regnell, and C. Wohlin, "Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment," *Empirical Software Engineering,* vol. 5, pp. 201-214, 2000.