Handshaking between Software Projects and Stakeholders Using Implementation Proposals

Samuel Fricker[1,2], Tony Gorschek[3], and Petri Myllyperkiö[4]

[1] ABB Switzerland Ltd., Corporate Research,
Segelhof, 5405 Baden-Daettwil, Switzerland
samuel.fricker@ch.abb.com

[2] University of Zurich, Department of Informatics,
Binzmuehlestrasse 14, 8057 Zurich, Switzerland
fricker@ifi.unizh.ch

[3] Blekinge Institute of Technology, School of Engineering,
PO Box 520, 372 25 Ronneby, Sweden
tony.gorschek@bth.se

[4] ABB Oy, Distribution Automation
B.O. Box 699, 65101 Vaasa, Finland
petri.myllyperkio@fi.abb.com

Abstract. Handshaking between product management and R&D is key to the success of product development projects. Traditional requirements engineering processes build on good quality requirements specifications, which typically are not achievable in practical circumstances, especially not in distributed development where daily communication cannot easily be achieved to support the understanding of the specification and tacit knowledge cannot easily be spread. Projects thus risk misunderstanding requirements and are likely to deliver inadequate solutions. This paper presents an approach that uses downstream engineering artifacts, design decisions, to improve upstream information, a project's requirements. During its preliminary validation, the approach yielded promising results. It is well suited for distributed software projects, where the negotiation on requirements and solution design need to be made explicit and potential problems and misunderstandings caught at early stages.

1 Introduction

Distributed multi-site 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 product development that is spread over continents and cultures. Distribution enables companies to leverage their resources and to draw on the advantage of proximity to customers and markets for large-scale product development [1].

The potential opportunities, however, also come with new challenges that affect both product management and product development of a company, and the requirements engineering of products. The threat of defect increase and cost overruns in multi-site development has been documented in literature and industry experience reports. Some of the main problems are attributed to heterogeneous understanding of requirements, and substantial differences in domain understanding and interpretation [2-4]. This is compounded by the fact that multi-site development usually is detrimen-

tal to informal communication between stakeholders, which include product managers, experts, and developers, as these roles are often separated geographically [2]. Informal communication and face-to-face meetings often help in augmenting imperfect specifications by building a common understanding of what is to be done, by whom, and when, and indirectly passing on domain knowledge and other tacit information crucial to the development effort. The ability for developers to seek out and regularly communicate with domain experts is prohibited by distance: all communication is associated with administrative and planning overhead, resulting in an raised threshold for daily validation of specification interpretations [3]. Cultural differences between sites can also lead to issues as some management styles prohibit developers from directly eliciting information: communication may be routed through one or a few central managers, further congesting communication [2].

A common result of these challenges is that defects, delays, and misunderstandings are caught very late, often during system integration. This dramatically increases the whole product development effort and is detrimental to time-to-market, which is recognized as one of the most important factors in market-driven development [5, 6].

In response to the challenges posed by distributed development this paper presents a technology developed in active collaboration with industry to alleviate some of the problems and enable explicit ***handshaking*** procedures between stakeholders. The technology, called ***implementation proposal***, enables such handshaking by relating software design to requirements. It was primarily motivated by challenges identified at ABB, and relates to a case where large scale development was performed utilizing sites spread across North America, Europe, and Asia.

Implementation proposals and their proper use enable explicit communication between stakeholders at the critical phase of requirements interpretation, as well as mapping the implications of design decisions to the end product. In addition, the comparison of implementation proposals and requirements demands iteration until a joint understanding of requirements and domain implications can be reached. A positive spin-off effect is that requirements deliverers, e.g. product managers, are able to gauge the impact on system architecture early in the process.

The focus of this paper is on presenting the implementation proposal technology and the organizational and process implications that follow the utilization of the technology. The experiences of using implementation proposals are based on a pilot currently underway in a large scale development effort at ABB.

The paper is structured as follows. Section 2 discusses the background and related work. In Section 3 the implementation proposal concept and handshaking process is presented and discussed. Section 4 presents early handshaking results. Section 5 positions handshaking with related literature. Section 6 concludes the paper.

2 Background

Large scale distributed development demands management of physical distance, time zones, and the thin spread of domain and technology expertise, which impact requirements communication [2], and management of the overall solution architecture with multiple levels of product integration.

Key principles that are applied to manage these issues include allocation of responsibility for well-separated components of the software solution to various teams, ownership of such a team for the overall lifecycle of their contribution, a globally accessible requirements and configuration management infrastructure, and project roles and practices that enable critical communication to happen among the teams as well as between the teams and product management, project management, and architects.

The application of these principles leads to an organizational structure that is aligned either with the structure of the software product and its related domains [7] or with the overall development process with different roles located at different places. Fig. 1 illustrates one such an alignment in a stylized and simplified manner that can be observed in ABB as it relates to the case presented in this paper.

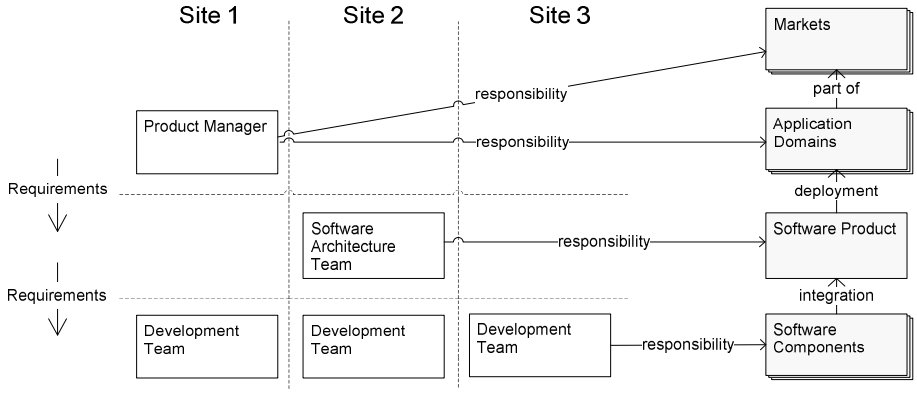

Fig. 1. Alignment of organizational structure with the structure of the software product and its related domains.

In Fig. 1, product management is responsible for a software product's markets and application domains and formulates relevant requirements, which are handed over to a software architecture team. The software architects, responsible for the overall architecture of the software product, communicate requirements to development teams, which are responsible for the development of the components assigned to them. Those components, finally, are integrated to form the software product, which after verification and validation gets deployed into the targeted application domains and markets.

There are several ways to handle the division of work and organizational structure with regards to distributed development [4, 7]. First, the case of ***hand-off*** can be seen in Fig.1, where different process steps are performed at different levels in the organization. Every such process step results in a deliverable that is handed down, like the requirements from product management to the architecture team. This implies that the deliverables have to be transferred across sites. Such a hand-off between sites can cause many of the issues discussed earlier in terms of heterogeneous understanding, and impossible compensation for imperfect deliverables due to lacking informal and day-to-day communication.

Second, the case of ***structural or functional division*** can be seen in Fig.1, where different parts of the product itself, i.e. some feature sets, are handled exclusively by one site. This implies that deliverables do not need to be transferred across sites, but are created and handed over locally. The main challenge is here to minimize coordina-

tion needs by a clear division of the product with low coupling between the parts that are distributed over sites. This is hard to achieve in practice.

In the ABB case hand-off challenges were predominant, even if some units on development level actually were organized according to product functions. The focus of this paper is on addressing the challenges to this type of distributed organization.

Looking at work performed previously in relation to the problem at hand, several investigations have been conducted for identifying the main challenges and recommending solutions [1-3, 5, 6]. Commonly recurring themes are face-to-face meetings and communication between sites. Solutions include introducing requirements management platforms for global access to requirements, employing communication technologies like chat, persistent video- and teleconferencing for enhanced communication, shared project workspaces, and configuration management systems.

A central issue was not only to address the problems of requirements understanding and communication, but also to find a technology that would enable explicit mapping of design decisions to the product requirements. Product management was the main author of requirements at early stages of the product development project and a central source of domain knowledge. However, the time available to product management for communicating requirements and for validating design decisions was limited. Thus, the communication between product management and the architecture team had to be explicit and concrete enough to avoid misunderstandings despite hand-off over sites, and efficient enough to make good use of time spent.

Traditional communication and face-to-face meetings are well established practices at ABB, as are the utilization of CASE tools over sites. However, the fundamental limitations of not being a team in one location demanded additional steps to be taken to ensure that a common understanding had been reached. One important goal was to increase the efficiency and effectiveness of the limited number of meetings by having relevant decision support material created beforehand as a part of the practices. This involved the creation of artefacts that would increase traceability between design decisions and requirements, the two main constituents of architectural impact.

3 Implementation Proposal Concept and Handshaking Process

It is well accepted that requirements are tightly linked to solution design. This holds for requirements and design decisions at any level of abstraction. This section elaborates on this relationship for the purposes of requirements communication and negotiation on an appropriate implementation approach by describing the structure and possible forms of implementation proposals and their relationships to requirements.

The relationship between requirements and solution design is bidirectional. Not only context and goals affect the design of a software solution, but also the emerging capabilities of the solution influences what goals can be achieved and how effective usage of the software shall be structured [8]. The impact of a targeted software solution on its context is particularly important to consider in situations with limited engineering resources, with limited capabilities of technology, and in projects that are building on legacy, as these factors pose demands on architecture and design regarding feasibility. The impact of a software solution is also important to consider, when

errors have been introduced in the design, due to imperfect understanding of requirements for example, which cannot be corrected with given project resources and deadlines, resulting in quality deficiencies and cost overruns.

3.1 Implementation Proposals

Implementation proposals support the negotiation between requirements and solution providers, as shown in Fig. 2. The requirements provider, a stakeholder or customer that is responsible for a problem domain, contracts a solution provider to realize a software solution. The solution provider, the supplier or development team, is responsible for creating a software solution that satisfies the requirements.

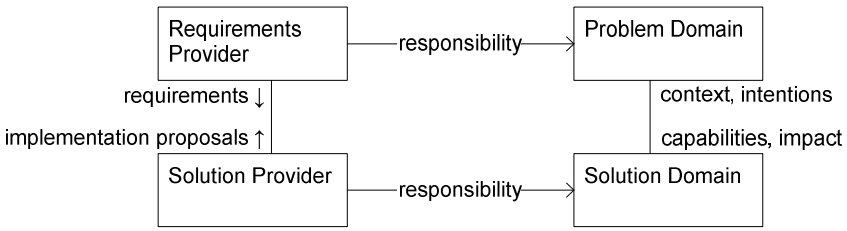

Fig. 2. Communication between requirements and solution providers through requirements and implementation proposals.

Solely focusing on requirements during negotiations is not enough as requirements are often misunderstood and the impact of feasible architecture and design is largely ignored. To mitigate these risks, implementation proposals are introduced. Implementation proposals describe the targeted solution and its expected impact from the perspective of the supplier. As Fig. 2 illustrates, implementation proposals are an answer to requirements and flow from the solution provider to the requirements provider.

The situation described in Fig. 2 appears often in product development at ABB. Referring to Fig. 1, the interaction pattern is of relevance between product management, the requirements provider, and the software architecture team, the solution provider, who need to coordinate the development of the overall software product. The pattern is also of relevance between the software architecture team, which now becomes the requirements provider, and every development team, the solution providers, that are responsible for the various software components. Not shown in Fig. 1 are the likely interactions between product management, the requirements provider, and some of the development teams, the solution providers, for coordinating lower-level requirements for design of externally visible software components.

Implementation Proposal Structure

A requirement describes a condition or capability needed by a stakeholder to solve a problem or achieve an objective [9]. To provide such information, typical requirement attributes, shown in Fig. 3, include a description of relevant context and assumptions (R1), the intention or goal to be achieved (R2), and the rationale behind the requirement (R3). Depending on the development process, these basic attributes are complemented with attributes covering the source of the requirement, the urgency and priority of the requirement, and others (R4+). While a requirement describes a problem to

be solved, it is considered good practice, not to describe any potential solution for solving the problem, for not to prematurely limiting the solution space.

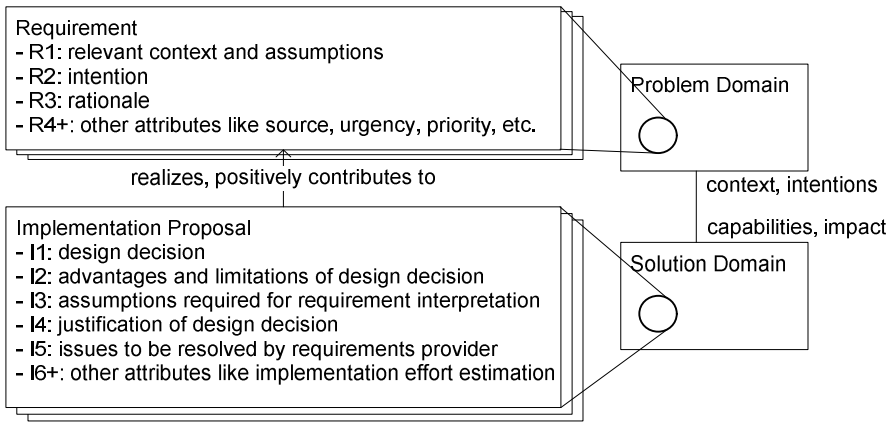

Fig. 3. Structure of and relationship between requirements and implementation proposals. R1 to R4+ are requirements attributes. I1 to I6+ are implementation proposal attributes.

To validate the understanding of a requirement and to set the right expectations on the solution that will be delivered, the supplier answers a requirement with an implementation proposal. As Fig. 3 illustrates, the implementation proposal needs to describe at least the *design decision* that is considered to satisfy the requirement (I1), and the effects of that design decision in terms of *advantages and limitations* (I2). These effects correspond to the inferred architectural impact of the decision on both the solution and the problem domains.

While the *design decision* and *advantages and limitations* attributes of the implementation proposal may be sufficient to document the results of the negotiation between stakeholder and supplier, they are often not enough to build a satisfactory level of trust between the parties that provided information has been correctly understood. To achieve such trust, two other attributes are introduced: *assumptions* used by the supplier for understanding what is meant with the requirement (I3) and a *justification* why the design decision is believed to be appropriate (I4).

The disclosure of *assumptions* for interpreting a requirement (I3) helps the two parties to manage the ambiguity that is inherent in human communication. Such ambiguity needs to be addressed in a particularly careful manner when the communication is made difficult, for example by physical distance or differing technological and domain background.

Justifying the design decision relates the implementation proposal to the broader context of the overall solution and problem (I4). The justification reveals why the supplier has chosen the particular design and not another one. It describes the trade-offs that have been made between relevant requirements that possibly stand in conflict with each other and limitations that were introduced by other design choices, including considered technologies. It is with this understanding that a customer can accept a design proposal that without such information may be considered sub-optimal.

A third type of attributes supports the management of the negotiation between requirements and solution providers. In early stages of requirements elaboration and

solution design, a lot of information necessary for deciding on an adequate design is lacking. To highlight such information needs, the solution provider describes the *issues that need to be resolved* to enable creating or improving the contents of the implementation proposal (I5). Such issues become a list of actions for the stakeholder who owns the requirement related to the implementation proposal. It is then through providing adequate context and rationale information that the stakeholder steers the evolution of the design. The negotiation on requirements and implementation proposals is considered to be concluded when all issues are resolved.

The last group of implementation proposal attributes covers information like estimation of implementation effort, implementation status, and other attributes that are specific to the chosen development process (I6+).

Relations between Implementation Proposals and Requirements

An implementation proposal describes how a given requirement is intended to be *realized* by a software solution. In some situations the design decision is not sufficient to conclusively address the requirement, in which case the relationship is said to *positively contribute to* realizing the requirement. These two relationships are indicated by the keywords *realize* and *positively contribute to* in Fig. 3.

Requirements and implementation proposals do not always stand in a one-to-one relationship, even-though many of them do so at the conclusion of the implementation. When requirements are handed over from the requirements provider to the solution provider, the initial set of requirements is without references to implementation proposals. Only as the solution provider's understanding of an appropriate implementation approach matures, implementation proposals are created.

At many stages of the design process, the requirements available to the solution provider turn out to be insufficient to make sound architectural decisions. In such situations it is not the requirement that comes first. Rather, an implementation proposal is used to elicit appropriate requirements. In this case the assumptions, justification, and issues attributes of the implementation proposal are of major importance to guide the stakeholders in providing the right kind of information and decision making.

An implementation proposal may positively contribute to multiple requirements. Such a constellation may express the advantages of a design decision [10]. However, it may also indicate a need for improving the implementation proposal: the implementation proposal not only defines what is intended to be implemented, but also how that design decision relates to the requirement (implementation proposal attributes I3 and I4). To improve the implementation proposal, the facets of the design decision specific to the individual requirements need to be highlighted. Then again, the situation may also indicate a need for improving the requirements: they may be overlapping or address similar concerns more effectively expressed by a single requirement.

Further improvement needs for requirements and implementation proposals may also be indicated by situations with one requirement affected by several implementation proposals.

- A requirement may be too abstract and needs to be refined into more detailed requirements, which are addressed by the individual implementation proposals.
- A requirement may not be sufficiently atomic and needs to be decomposed into its aggregated parts that are addressed by the individual implementation proposals.

- There may be a number of design options to satisfy a requirement. Every option is proposed as an implementation proposal and it is up to the requirements provider to select, which of the options shall be chosen, if not all.

These constellations of how implementation proposals relate to requirements can pinpoint various kinds of potential defects. Still, they are not a call for driving unnecessary formality. Rather, the discussed constellations are useful to support the handshaking parties in enhancing their communication by triggering actions such as improving information. The consideration of these constellations complements the use of the implementation proposal attribute 'issues to be resolved' (I5).

The interaction between the two parties, the requirements provider and the solution provider, supports the continuous improvement of the quality of both, requirements and implementation proposals. The responsibility for contributing one's part to project success leads to a continuous mutual pull for increased quality of information. While such a pull may be observed in a majority of projects, implementation proposals make the status of information and the need for information improvement explicit, thus manageable. Also, a learning effect can originate from such collaboration: learning how to write requirements and implementation proposals that are understandable and useful for the other party. Such quality improvement and learning has been observed, for example, when testers have been involved in reviewing specifications [11].

Forms of Implementation Proposals

The description of design decisions may take different forms and levels of detail, depending on whether high-level architecture or detailed design is captured, depending on how understanding or feasibility risks need to be addressed, and depending on the CASE tool infrastructure in the software company.

Implementation proposals may be formulated with tools that are used for requirements management. The attributes suggested for describing implementation proposals are outlined in Fig. 3. The writing style should be short and concise so that the formulation of the implementation proposals does not take unnecessary time.

While a majority of implementation proposals are simple to convey, a few require considerable elaboration. In this situation, documents are written whose structure corresponds to the implementation proposals attributes. These documents are then attached to the entries in the requirements management database.

Companies that adopted a model-driven development approach [12] may want to formulate implementation proposals as part of their software model in a semi-formal graphical language like UML [13]. The company may choose not only to document the design decisions in such a language, but also complete implementation proposals. This works well if the requirements are documented as part of the model.

While a text-based or semi-formal documentation approach is useful for some classes of requirements, others are easier to answer with prototypes. Usability requirements, for example, may lead to implementation proposals that capture the design decision in form of a graphical user interface prototype or mock-up.

The goal of implementation proposals is not to prescribe form, but to support the interaction and negotiation between requirements and solution providers. Decisions about formality and methodology should be taken by the involved parties by considering situational contingencies to maximize efficiency and yield of communication.

3.2 Handshaking Process

To achieve an understanding between a requirements and a solution provider and to agree on requirements and the intended solution, the two parties follow a handshaking process that spans three phases as illustrated in Fig. 4.

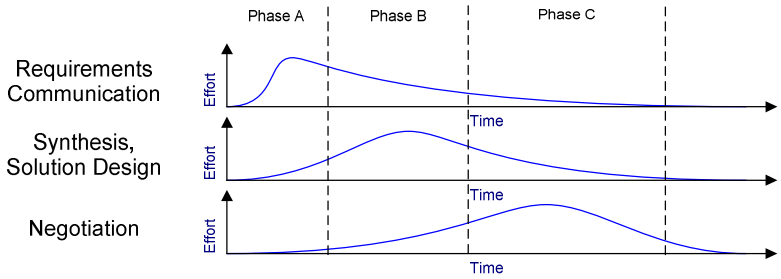

Fig. 4. Overlapping handshaking activities: requirements communication, solution synthesis and design, and negotiation using implementation proposals. The three phases A, B, and C represent time spans with different focus with respect to handshaking activities.

Phase A connects the handshaking process with the requirements management processes performed by the requirements provider [14] and the problem domain understanding that the requirements provider has already established prior to the handshaking process. Initial requirements and information about the problem domain are communicated to the solution provider. This set of requirements represents the starting point for the work of the solution provider. It typically does not satisfy desired qualities of requirements specifications like unambiguity and completeness [15, 16].

During *Phase B*, the requirements receiver synthesizes the received problem domain data and technology knowledge to identify implementation approaches that would satisfy the requirements. The process of such synthesis is highly complex and closely related to the experience of the designing people [17].

Phase C aims at achieving an agreement on the intended realization of the solution. It is the central phase, where implementation proposals are used to validate the solution provider's understanding of requirements, to improve the requirements, and to validate the adequacy of the intended solution. The goals of the negotiation activities shift over time. The later the negotiation activities are, the less likely they are to modify the design, but to correct the understanding of achievable product capabilities and their impact.

Data Flow

Fig. 5 illustrates how the requirements and the solution providers interact with each other by describing the dataflow between their activities and information repositories. The requirements communication and solution design processes from Fig. 4 are shown in Fig. 5 without modification. The negotiation process covers all four activities in Fig. 5, requirements communication, solution design, and implementation proposal formulation and review, all of which are performed iteratively. The handshaking process assumes that the two parties share requirements and implementation proposal data.

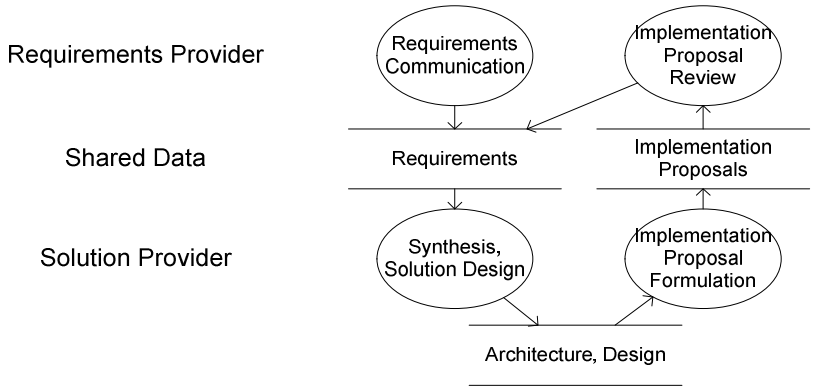

Fig. 5. Dataflow between handshaking activities and information repositories.

The requirements provider communicates requirements, which are used for solution design by the solution provider. During negotiation, the solution provider formulates implementation proposals that are based on that design, which are then reviewed by the requirements provider. Reviews of implementation proposals focus on whether the intended solution makes sense with regard to the requirements provider's interpretation of requirements. Review comments then lead to requirement improvements by the requirements provider and to subsequent changes to solution design and implementation proposals by the solution provider.

Some of the design decisions that need to be taken by the solution provider are not foreseeable by the requirements provider. As a consequence, insufficient information for guiding these design decisions is provided during requirements communication. Here, the solution provider elicits relevant information by submitting implementation proposals for review that are not connected to requirements initially, but which are complemented with requirements as a result of the implementation proposal reviews. In this case the implementation proposals drive the elicitation of requirements.

Success Criteria

Commonly used criteria for evaluating the quality of requirements specifications in a traditional unidirectional requirements communication context include completeness, ambiguity, correctness, and consistency among others [18]. The success of handshaking using implementation proposals can be evaluated with the same criteria, but requires a new interpretation of these criteria. For example, some of these qualities are achieved as an inherent capability of the handshaking process, while others can be evaluated more comprehensively because additional information is available.

Completeness is not only evaluated by considering completeness of the requirements with respect to goals and coverage of the problem domain, but also by asking:

- Are the implementation proposals covering all requirements?
- Are the implementation proposals sufficiently covering the intended solution?
- Are the requirements covering all implementation proposals?

The management of requirements *ambiguity* is a fundamental capability of the handshaking process. A requirement can be considered understood by the requirements receiver, when it is covered by at least one accepted implementation proposal.

Correctness of requirements in the sense of correctly describing the desires and needs of stakeholders and of correctly describing the properties of the problem domain is not affected by the handshaking process and needs to be ensured by traditional requirements engineering techniques. *Feasibility of requirements* and *correctness of architecture and design*, however, is guaranteed to a large extent when requirements and implementation proposals match. Nevertheless, such correctness holds only to the degree as the belief is correct that the intended solution actually yields the capabilities and impact that are described by the implementation proposals [17].

Consistency of requirements is evaluated in the handshaking process by the solution design activities. Handshaking also introduces additional consistency needs:

- Are the implementation proposals consistent among themselves?
- Are the implementation proposals consistent with the intended solution?
- Are the implementation proposals consistent with the requirements?

Evaluation of the latter, consistency between requirements and implementation proposals, is an essential part of the review activities performed by the requirements provider during negotiation. Achieving the former two consistency needs depends on the practices of the solution provider.

Successful requirements engineering does not only depend on the quality of information that is produced, but also on stakeholder satisfaction and commitment. Implementation proposals must help to set appropriate expectations on the targeted solution, inform about required changes in the problem domain, and ensure that the problem domain changes are feasible and fit within the strategic orientation of the requirements provider, thus making it possible to defend the chosen solution [19].

While much of these requirements engineering services is not explicitly captured in the implementation proposal structure and handshaking process, relevant knowledge and understanding emerges out of the focussed interaction between the requirements and solution providers. Understanding is attained and expectations are set not only by discussing requirements, but also by examining the intended solution and how it addresses the requirements. The reviews of implementation proposals, performed by the requirements provider as part of the negotiation phase, ensure that required changes in the problem domain are known, feasible, and aligned with strategy.

4 Preliminary Experiences

The handshaking process using implementation proposals has come out of an industrial need to manage the handover of requirements to a distant project team. This section describes some preliminary experiences with handshaking and what its potential advantages and limitations are. While scientific validation of the implementation proposal concept and handshaking process is part of ongoing research, this section illustrates how the approach is used in a broader context.

The handshaking process was established in a globally distributed project that involved about 50 engineers and that was structured according to Fig. 1. The project was organized according to a toll-gate model [20]. Important toll-gates included the following ones [21] and mapped to the handshaking phases (Fig. 4) as follows:

- *Agree to start project*: start of Phase A

- *Agree on requirements and project plan*: end of Phase B
- *Agree on release*: end of Phase C[1]

The toll-gate *agree on requirements and project plan* is interesting to study for understanding the use of handshaking in a complex product development scenario. This toll-gate assumes that high-level architecture is defined and satisfies important requirements. Thus, the interface between product management and the product architecture team has reached the end of Phase B. The interface between the product architecture team and the individual development teams, however, may not have progressed so far yet, which yields similarities with concurrent engineering [22].

The timing of the toll-gates was fixed for the project. This implied a time-box-oriented approach to achieving the goals of the project phases. For example, requirements were not perfect at the toll-gate *agree on requirements and project plan*, but the best-possible quality within given time and resources.

Product-level handshaking was achieved with implementation proposals integrated into a requirements management infrastructure. Both requirements and implementation proposals were captured in tabular form. Upon need, an explicating document was created and attached to the implementation proposal.

Handshaking between product management and components with product-external interfaces was mostly performed using prototypes. Prototype validation leads to complemented requirements and subsequent modification of component design.

Handshaking between product architecture and components was not considered in this preliminary experience. The results that were achieved with product-level handshaking encouraged the architects to pilot the concept, however.

Negotiation activities typically were performed in meetings. These meetings were used for discussing requirements and implementation proposals and for making decisions. Pure text-based communication was less frequent. Text was used to document the information gathered and decisions taken during the negotiation meetings in the requirements management database. Thus, work with implementation proposal is not a continuous process as Fig. 4 might suggest, but peaked where meetings took place.

Comparing the early experiences of using implementation proposals with the former requirements hand-over approach, the *product manager* elaborated:

- Agreement on requirements with the architects was usually not a problem. However, there were usually problems in understanding the impact of the requirements on the architecture, which led to unacceptable software architectures. It is important to establish trust between product management and software development. Implementation proposals help to see how requirements are realized before an inadequate solution is chosen, which is difficult to change.
- Handshaking work is more structured. The implementation proposals are usually discussed in meetings and are then used as a means to make decisions and as a form of documenting these decisions.
- Implementation proposals are most useful in areas where risk is high.

Software architects mentioned:

[1] At the time of writing, the project had passed tollgate *agree on requirements and project plan*. Phases A and B were observed and phase C planned.

- Requirements are often too fragmentary to build sound software architecture. Implementation proposals help us to highlight important design decisions, where input is needed from the product manager. Only when requirements and implementation proposals are completed, the toll-gate 'agree on requirements and project plan' should be passed.
- The software architecture is dependent on inputs from many product managers. Design decisions are not only influenced by one product manager, but need to account for the needs of others and for the architecture of the surrounding system.[2]
- It is important to allow implementation proposals be described in different forms such as entries in the requirements management software, as architectural documents, and as prototypes.

The project changed from uni-directional communication of requirements to handshaking with implementation proposals, which led to early discovery of problems, which would have been discovered only at solution validation late in the development process. Based on this experience, project members estimated a return on investment between ten and fifty times the cost of the process change due to risk reduction.

Clearly, the preliminary experiences confirm the industrial need for improved handshaking procedures. Implementation proposal-based handshaking fits well into practical industrial distributed development and has lead to encouraging results. Still, while managing ambiguity to improve the level of trust and managing the handshaking process are perceived important and are lived by the practitioners, they are not perceived as the silver bullet. In particular it needs to be studied how multiple stakeholders can be addressed and what activities should accompany the use of implementation proposals to further support increase the appropriateness of a software solution.

5 Related Work

The challenge of correctly understanding requirements has already been addressed by iterative development processes [23]. Such a process aims at reducing the risk of costly rework by shortening the development cycle and allowing validation of partial work results. In principle, such a process implements a feedback paradigm [24], where the customer is the goal-defining element and the project team the goal-implementing element whose outputs need to be controlled.

Handshaking using implementation proposals builds on a similar feedback mechanism. Handshaking, however, poses fewer requirements on the engineering results for validation and is more focussed on the interface between customer and supplier.

In addition to partially implemented solutions that result from a full iteration, handshaking accepts early work results such as design decisions, models, and prototypes that result from solution analysis and design activities. This allows detecting errors earlier and makes such detection independent of the development process, hence also supporting sequential software development scenarios.

The information that is fed back during handshaking is a special form of design rationale [25]. In contrast to other design rationale approaches, handshaking aims at

[2] Note that such a scenario has not been discussed in this paper.

ensuring that the solution provider's intended results corresponds to the expectations of the requirements provider, while establishing an atmosphere of trust. The design rationale information consists here of requirements, design decisions and implementation proposals, which carry the necessary information to relate design decisions to requirements. The notation for capturing the design rationale is intentionally left open for adapting to domain-specific practices and development context.

6 Conclusions and Future Work

Implementation proposals contribute to a better understanding of requirements. Focussing on the interface between a stakeholder like a product manager and a development team, the explicit description of design decisions and their impact on requirements helps the stakeholder to understand and adjust what the development team will build.

While not using explicitly documented implementation proposals may be sufficient for projects with collocated development teams and stakeholders, written information exchange must be enhanced in a distributed setting to build trust, and manage the ongoing negotiations. Implementation proposals help achieve these goals by relating requirements to design decisions, uncovering assumptions in the interpretation of requirements, justifying design decisions, and highlighting issues to be resolved.

The use of implementation proposals, in addition to the obvious, also has positive spin-off effects which can result in improved quality and catching of defects earlier in the development process. Creating improved decision support material early in the project process can vastly improve the accuracy of estimation and risk analysis. These are especially important in market-driven development as time to market is crucial.

The cost of creating implementation proposals may be seen as a drawback, although it should be realized that the artefacts themselves, both better requirements and the design decisions captured by the implementation proposals, can be reused as decision support material, design material, and bases for system test activities, effectively spreading the cost over several development phases. In addition, as experience in using implementation proposals increases, the maturity of the distributed product development environment grows. This makes it possible to create less formal artefacts as domain and technical understanding becomes more homogenous across the teams. The learning effect resulting from using implementation proposals not only spreads domain and technical knowledge, but also supports product management in detecting defects in requirements. Ultimately, better requirements can be written from the start.

Future research will focus on empirically validating the implementation proposal concept for requirements handshaking in distributed software development contexts. The yield and usability factors of the implementation proposal concept shall be investigated and compared it with traditional approaches for requirements communication. Also, the implementation proposal concept will benefit from further development by studying how requirements and solution design interact over multiple levels of abstraction and by considering more than a single requirements provider.

References

1. Battin, R.D., Crocker, R., Kreidler, J., Subramanian, K.: Leveraging Resources in Global Software Development. IEEE Software **18** (2001) 70-77
2. Damian, D., Zowghi, D.: RE Challenges in Multi-Site Software Development Organisations. Requirements Engineering **8** (2003) 149-160
3. Herbsleb, J.D., Paulish, D., Bass, M.: Global Software Development at Siemens: Experience from Nine Projects. 27th International Conference on Software Engineering. ACM, St. Louis MO (2005)
4. Herbsleb, J.D., Mockus, A.: An Empirical Study of Speed and Communication in Globally Distributed Software Development. IEEE Transactions on Software Engineering **29** (2003) 481-494
5. Dahlstedt, A., Karlsson, L., Persson, A., NattochDag, J., Regnell, B.: Market-Driven Requirements Engineering Processes for Software Products – a Report on Current Practices. International Workshop on COTS and Product Software RECOTS, Los Alamitos CA (2003)
6. Regnell, B., Beremark, P., Eklundh, O.: A Market-Driven Requirements Engineering Process - Results from an Industrial Process Improvement Program. Requirements Engineering **3** (1998) 121-129
7. Herbsleb, J.D., Grinter, R.E.: Architectures, Coordination, and Distance: Conway's Law and Beyond. IEEE Software **16** (1999) 63-71
8. Jackson, M.J.: Software Requirements & Specifications: a Lexicon of Practice, Principles, and Prejudices. Addison-Wesley Pub. Co., New York Wokingham, England ; Reading, Mass. (1995)
9. IEEE Computer Society. Standards Coordinating Committee.: IEEE Standard Computer Dictionary: a Compilation of IEEE Standard Computer Glossaries, 610. New York, NY, USA (1990)
10. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic, Boston MA (2000)
11. Gorschek, T., Dzamashvili - Fogelström, N.: Test-case Driven Inspection of Pre-project Requirements - Process Proposal and Industry Experience Report. Requirements Engineering Decision Support Workshop, Paris (2005)
12. Stahl, T., Völter, M.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley, Chichester, England ; Hoboken, NJ (2006)
13. Object Management Group, "Unified Modeling Language (UML), Version 2.0", 2005
14. Gorschek, T., Wohlin, C.: Requirements Abstraction Model. Requirements Engineering Journal **11** (2006) 79-101
15. IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998
16. Fricker, S., Glinz, M., Kolb, P.: Case Study on Overcoming the Requirements Tar Pit. Journal of Universal Knowledge Management **1** (2006) 85-98
17. Kruchten, P.: Casting Software Design in the Function-Behavior-Structure Framework. IEEE Software **22** (2005) 52-58
18. IEEE: Recommended Practice for Software Requirements Specifications (Standard 830-1984). IEEE Press, New York NY (1984)
19. El Emam, K., Madhavji, N.H.: Measuring the Success of Requirements Engineering Processes. 2nd IEEE International Symposium on Requirements Engineering, Los Alamitos, CA. (1995)
20. Cooper, R.G.: Winning at New Products: Accelerating the Process from Idea to Launch. Perseus Pub., Cambridge, Mass. (2001)
21. Wallin, C., Ekdahl, F., Larsson, S.: Integrating Business and Software Development Models. IEEE Software **19** (2002) 28-33
22. Davis, A., Sitaram, P.: A Concurrent Process Model of Software Development. ACM SIGSOFT Software Engineering Notes 19:2 (1994) 38-51
23. Pressman, R.: Software Engineering: A Practitioner’s Approach. McGraw-Hill. (2004)
24. Klir, G.: Facets of Systems Science. Springer. (2006)
25. Moran, T., Carroll, J.: Design Rationale: Concepts, Techniques, and Use. Lawrence Erlbaum Associates. (1996)