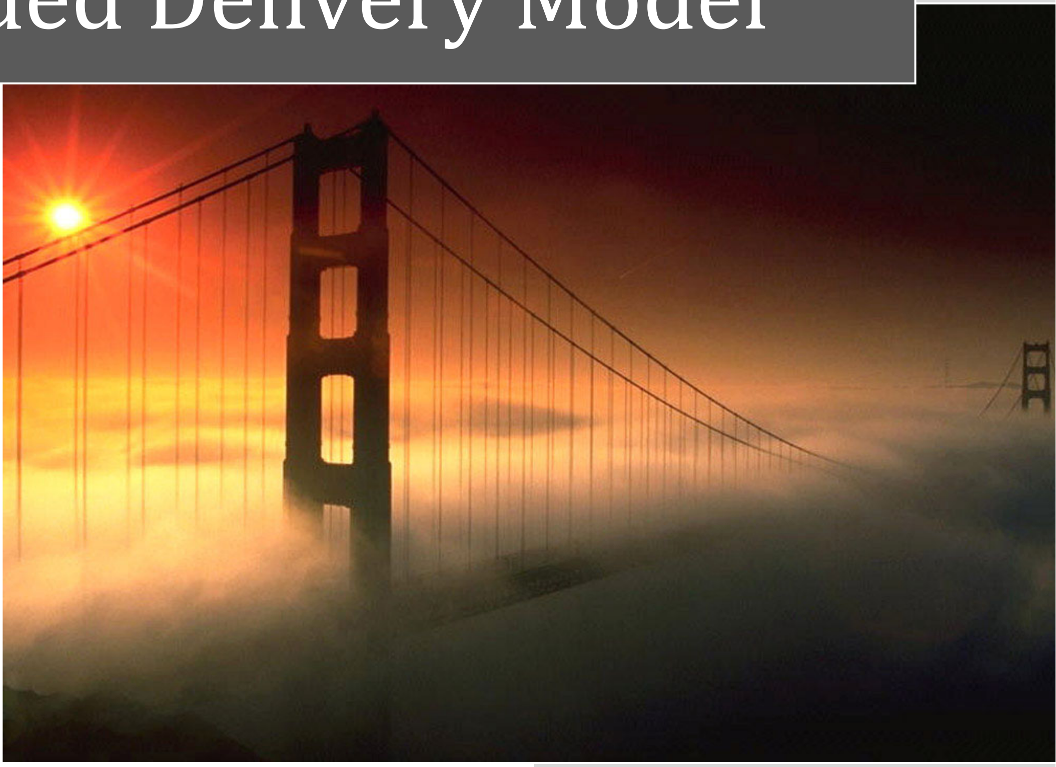
Alignment of Requirements & Architectural Design In a Blended Delivery Model



Reinier Kernkamp

University of Twente
Enschede - The Netherlands

logicaCMG

Releasing your potential

Alignment of Requirements & Architectural Design In a Blended Delivery Model

Author:	Reinier Kernkamp
University:	University of Twente
Master:	Business Information Technology
Track:	ICT & Innovation
Project:	Alignment of Requirements and Architectural Design In a Blended Delivery Model
Internal supervisors:	Dr. M.L. Ponisio
	Dr. A.B.J.M Wijnhoven
External supervisors:	P.A. Vruggink, MSc
	Ing. R.D.A. Jourdain

Arnhem, September 25, 2007

Management Summary

Requirement and architectural alignment is the process that determines the optimal fit between requirement and architectural design. At LogicaCMG, there currently is little understanding about how to structure this process during an outsourcing project in India. In such a global software development project, there is lack of alignment between the part where the analyst specifies the requirements and the part where the designer makes the design decision. This was the reason to explore the following research question:

"What improvements can be made to the RUP development model of a multi-department software factory, which results in better alignment of requirements and architectural design in a blended delivery model?"

The answer to the research question will explain what alignment concepts are relevant in this particular case and what enhancements the multi-department software factory should add to their current development model. The scope of this research is limited to only the *"Requirement"* and *"Analysis & Design"* disciplines in the RUP development process framework of LogicaCMG. To get to know the important alignment concepts, an explorative research was chosen. From theory different alignment concepts were identified by looking at communication, team dynamics, requirements specification styles and architectural design concepts. The research showed fifteen important alignment concepts, which have altered and validated by means of four in depth case studies. The following recommendations can be made:

- General guidelines; make use of gatekeepers to communicate functional and technical design, bring the Indian lead developer onshore for knowledge exchange, give training in English, RUP and commonly used components, take Indian holidays into account in project plan, frequently use communication tools and give cultural training about how to communicate with each other.
- Requirements guidelines; always specify Vision, Use Case Model, Supplementary Specifications, Glossary and a User Interface document, review requirement documents with tester, builder and customer, define relevant nouns in Glossary document, let requirements specifiers translate the artifacts themselves into English, make use of proper requirements management tooling and make use of preliminary version of the product or prototype to identify wrong User Interface requirements.
- Analysis & Design guidelines; only add contextual information in documents if tasks are non-routine, at least a detailed design of Software Architecture Document, Data Model and Analysis model is needed in development location, use well known and properly documented building blocks, make use of architect located in India when system is complex, exchange knowledge using design and analysis patterns, thoroughly specify functions and realistic estimates of the system using metrics in close cooperation with customer, clearly specify interfaces of the system and the configuration of the system environment.

The guidelines can be placed into the current guidelines section of the existing RUP development model of Result Centre with little effort. Embedding the recommendations in the current RUP development framework provides future analysts, designers and project managers guidelines to reduce the design problems during global software development. This does not only limit the risks and costs during a project, but is ultimately going to make it more likely that the system under development meets the customer's expectations.

PREFACE

First of March, I started the journey to discover the world of aligning requirements with design to obtain the title Master of Science. Personally, I found it quite interesting to get more knowledge about this area of research that is becoming more and more important in the real world. This master's thesis is the final part of the Business Information Technology (BIT) curriculum at the University of Twente in Enschede. The actual research project is done at a multi-department software factory of the IT-service provider LogicaCMG.

Looking at my study BIT, it was a topic concerning process innovation, which is one of the key specializations during my study. In my opinion, it is a "perfect match". Fortunately, I can make a difference by bringing two different worlds together; that is the link between theory and practice.

The results of this report would not have been possible without all the support from the LogicaCMG people and I would like to thank everybody that helped me during my research project. Availability sure can be a bottleneck during a research project and it is amazing to see that people working on high demanding projects could help me out by letting me interview them and helping me to get information from different sources. In addition, there was good guidance and support from my supervisors from the University of Twente, as well as from LogicaCMG.

First of all, I would like to thank the supervisors from the University of Twente, who really helped me out to get a good abstract framework, which has been used as one of the fundaments of my research. Laura Ponisio, my first supervisor was a great inspirator and really motivated me to get the work done within the six months. Her reviews and suggestions helped me to keep on the right track and she was like a catalyst giving me a boost when I was stuck at particular points of my research. Fons Wijnhoven provided me with good pointers about the things that are required to be in my thesis and helped me with structuring my research by introducing the different communication dimensions and related theory.

Besides support from the University, the supervisors of LogicaCMG really supported me during my research. They gave me access to documents, papers, persons and other resources I needed for my research. I would like to thank Peter Vruggink, who provided me with several contacts for my case studies and assisted me with several decisions during my research. Raynni Jourdain provided me with all facilities to properly do my work and also helped me to contact the respondents for my last two case studies. I was kind of stuck at that moment and I was really glad he helped me out there. In addition, I would like to thank all Working Tomorrow students located in Arnhem, Peter Vruggink and Raynni Jourdain for the great working environment. As a final point, I would like to thank my parents for supporting me the past six months.

Arnhem, September 2007

Reinier Kernkamp

Brief Table of Contents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 RESEARCH SETTING

This research takes place at LogicaCMG in The Netherlands, Arnhem. LogicaCMG is providing IT services at international level. Around 40,000 people across 41 countries are working for LogicaCMG and in 2006, the company generated a revenue of £2,665.2 million. LogicaCMG focuses on enabling its customers to build and maintain leadership positions using the industry specific knowledge and delivering its track record for successful delivery. Their main activities are: providing business consulting, systems integration and IT and business process outsourcing across diverse markets including telecoms, financial services, energy and utilities, industry, distribution and transport and the public sector.

LogicaCMG has six business units, as can be seen in picture 1. This research project is done for the business unit Public Sector. Furthermore, this research takes place at the Result Centre, which is a software factory for multiple departments of LogicaCMG. The Result centre offers:

1. application development,
2. application management,
3. facilities to develop own software for system development and application management projects for internal and external clients,
4. advice about setting up, manage and assess software of software factories of customers.

The Result Centre is LogicaCMG's software development delivery model. The mission of the Result Centre is to develop quality software in a predictable way, in less time and with lower costs: better, cheaper, faster, predictable. The development and maintenance process is measured using CMMI (CMMI Product Team 2006). In India, LogicaCMG has maturity level 5 and in the Netherlands they want to be at maturity level 3. Ultimately the goal is to be at the same maturity level as in India.

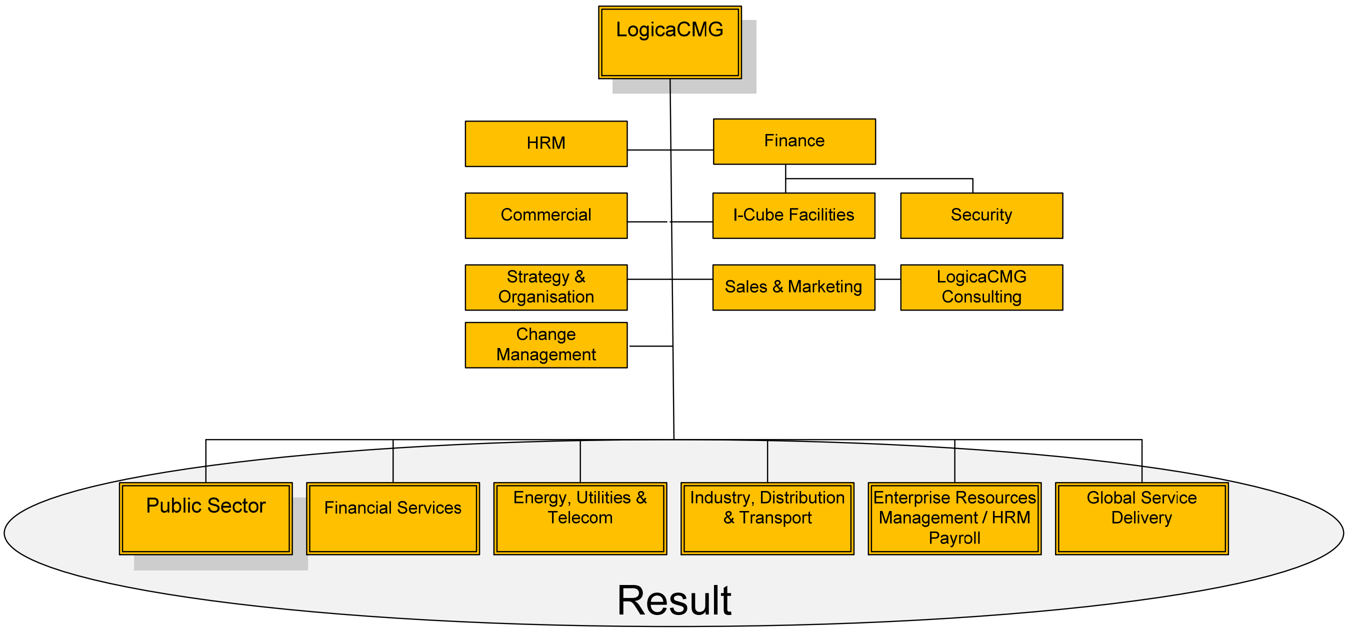


Figure 1 - Organization structure; the 6 branches below are business units

1.2 PROBLEM STATEMENT

During system development requirements tend to frequently change and are sometimes misinterpreted by different stakeholders. A system design is sometimes not capable of adapting new requirements quick enough due to organizational and technological factors. In the early phases, requirements are established based on analysis of business goals and analysis of the application domain. During the transition from requirements to architecture, designers do not always use the requirements specification directly and it appears that these two phases are poorly integrated. LogicaCMG is currently offshoring development of large software systems to India and alignment of requirements and design is becoming more and more important for them.

1.2.1 Blended Delivery Model

For LogicaCMG, the blended delivery model is an approach to make use of globally available resources. This model is part of LogicaCMG's Global Software Delivery business and operates through multi-discipline centers of excellence where LogicaCMG tries to make use of the most effective and efficient geographical locations. There are multiple tiers, which can be onsite (at the customer's office), offsite (off the customer's premises), nearshore (Europe) and offshore (outside Europe).

This four-tiered approach can be seen in figure 2. The blended delivery model makes it possible to access the best resources all over the world. In figure 2, the different types of sources are depicted as ellipses and can be found on the left. The advantages of this approach are bulleted in the rectangle on the right. (LogicaCMG, Website last accessed May 27th 2007)

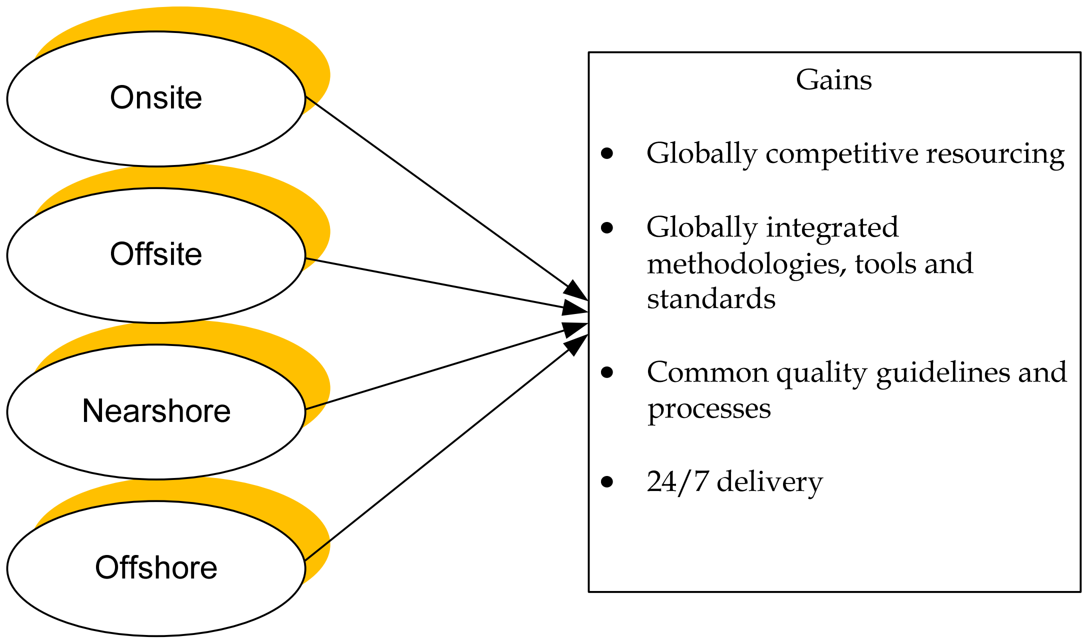

Onsite
Offsite
Nearshore
Offshore
Gains
• Globally competitive resourcing
• Globally integrated methodologies, tools and standards
• Common quality guidelines and processes
• 24/7 delivery

Figure 2 - Blended Delivery Model of LogicaCMG

Figure 3 shows when the decision at LogicaCMG is made to do offshore, onshore or nearshore development. If the complexity is high and there is high customer interaction, the development is mostly done onshore or nearshore. With low customer interaction and low complexity and risk, offshoring is frequently used. (LogicaCMG 2007)

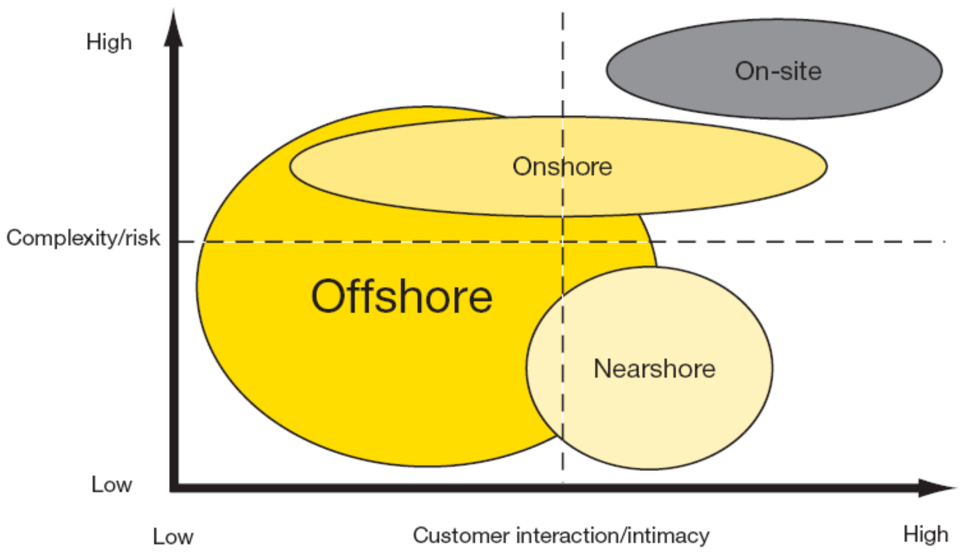


Figure 3 - Deciding where to do development (LogicaCMG 2007)

The blended delivery model means that part of the product life cycle is done at different locations. Therefore, proper communication and alignment between analysts and architects is crucial during system development in order to be able to make the intended system. Figure 4 shows the relation of the requirements and the different phases in an IT-project. Nearshore activities are done in countries within Europe and offshore activities are mainly done in India.

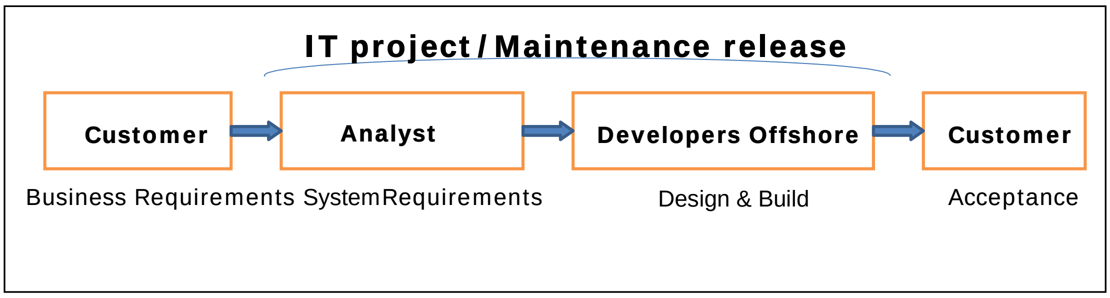


Figure 4 - Requirements from customer to development and back again

1.2.2 Stakeholders

There are different stakeholders in different layers, figure 5 shows the relationships between them in the different zones (Alexander, I. and Robertson, S. 2004). This gives us a good abstract model for the stakeholders involved. In the Onion Model four circles can be identified:

1. “The Product” or “The Kit” that is being developed.

2. “Our System” consists of “The Product” plus its human Operators and the standard operating procedures or rules governing its operation.

3. “The Containing System” includes “Our System” plus any human Beneficiaries of Our System.

4. “The Wider Environment” contains “The Containing System” and any other Stakeholders.

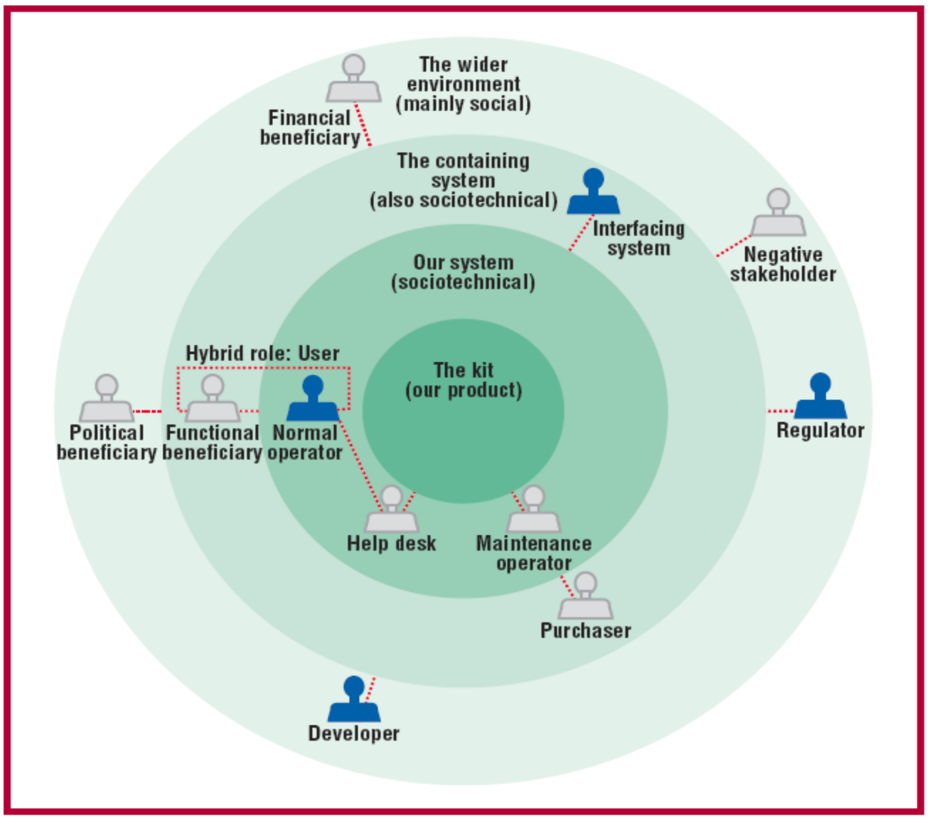


Figure 5 - Stakeholder relationships (Alexander, I. and Robertson, S. 2004)

For the LogicaCMG there are many stakeholders that can be identified. For this research only the requirements engineers and system developers are important, as they are the ones that benefit directly from tackling the main problem.

At Result Centre, RUP is mainly used for software development, which means that specific role sets are used, as can be seen in table 1.

Table 1 - Role sets in RUP (Kruchten, P. 2003)

Manager	This role is primarily involved in managing and configuring the software engineering process.
Analyst	An analyst is primarily involved in eliciting and investigating requirements.
Developer	These are the persons who are designing and implementing software.
Tester	This is the role that primarily deals with in testing the software.
Production and Support	Roles needed in order to support the software development process, or in order to produce additional materials required by the final product.
General	The roles that do not fit into one of the other role sets. An example is *reviewer,* which provides timely feedback to project team members on the work products.

Indirect stakeholders are: the company paying for the product (Purchaser), the Customer that uses the product (User), the Testers, the Production and Support personnel and of course also the people investigating this area of research.

1.2.3 *Alignment Problems*

This paragraph starts with explaining the main problem of the RUP development model that is being used at the Result Centre of LogicaCMG. Furthermore, different alignment problems will be listed and a final problem bundle will be drawn that shows the relationship between all the different alignment problems.

The main problem for the Result RUP model is:

The lack of alignment between the customer's and developer's site in a multi-department software factory with global software development activities.

In this statement, the customer's site is the part where the analyst specifies the requirements for the client and developer's site is the part where the designer makes the design decisions.

There are several problems that cause the lack of alignment between requirements and design. Figure 6 shows the problem bundle (Wieringa, R.J. and Heerkens, H. 2004) concerning the causes and impacts of our main problem in the blended delivery model.

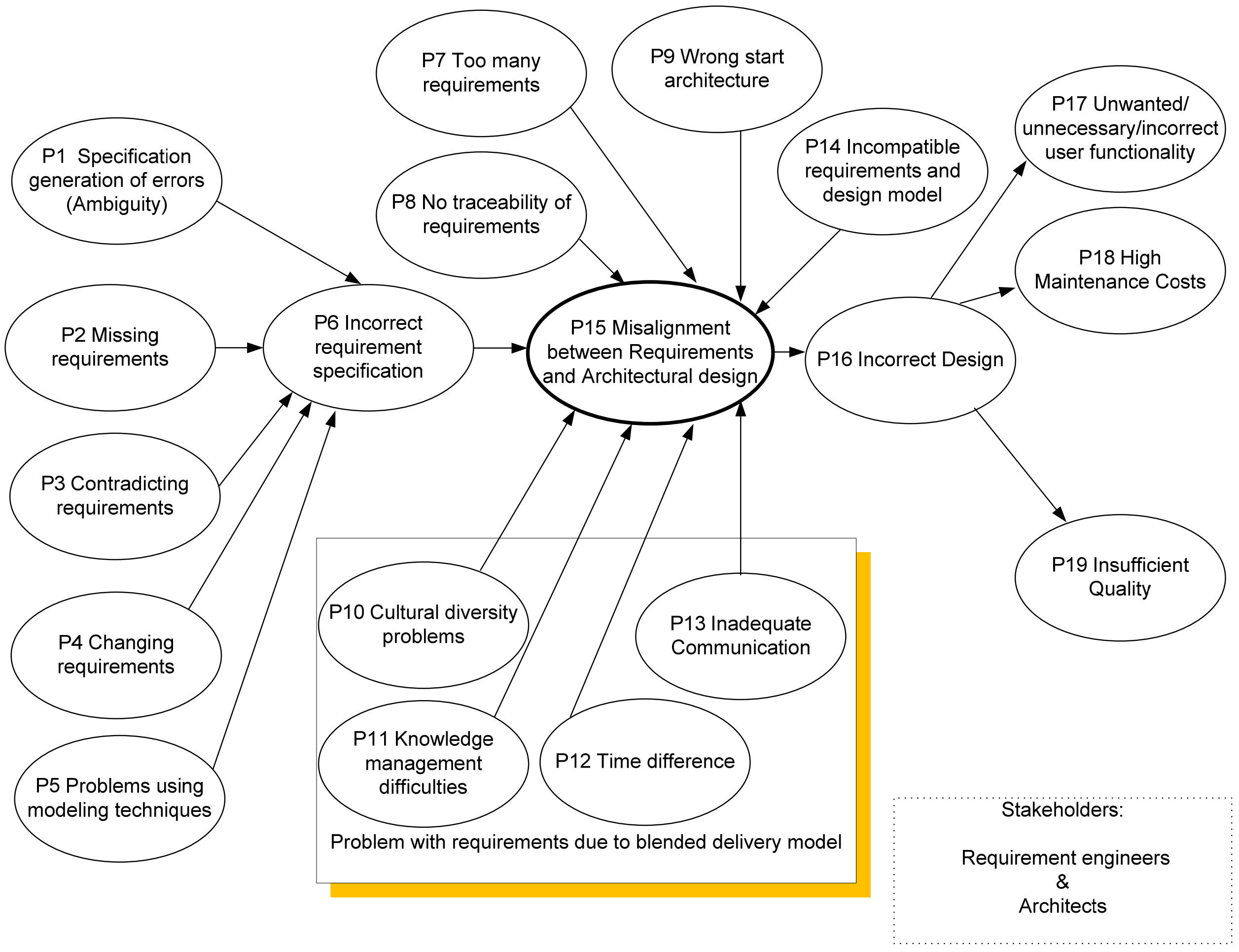

P7 Too many requirements
P9 Wrong start architecture
P17 Unwanted/ unnecessary/incorrect user functionality
P1 Specification generation of errors (Ambiguity)
P8 No traceability of requirements
P14 Incompatible requirements and design model
P18 High Maintenance Costs
P2 Missing requirements
P6 Incorrect requirement specification
P15 Misalignment between Requirements and Architectural design
P16 Incorrect Design
P3 Contradicting requirements
P10 Cultural diversity problems
P13 Inadequate Communication
P19 Insufficient Quality
P4 Changing requirements
P11 Knowledge management difficulties
P12 Time difference
P5 Problems using modeling techniques
Problem with requirements due to blended delivery model
Stakeholders:
Requirement engineers
&
Architects

Figure 6 - Problem bundle Requirement and Design

There are two kinds of problems, which are knowledge problems and world problems. A world problem is defined as follows (Wieringa, R.J. and Heerkens, H. 2006):

"World problems consist of a difference between the way the world is and the way we think it should be."

The next definition explains a knowledge problem (Wieringa, R.J. and Heerkens, H. 2006):

"Knowledge problems consist of a lack of knowledge about the world. To solve a knowledge problem, we need to change the state of our knowledge, and when we do that, we try not to change the world."

The main problem that is going to be researched is a knowledge problem, as there is little understanding about alignment of requirements and architectural design during global software development.

1.2.3.1 Requirement specification errors & user functionality issues

Looking at requirement defects, (Kosman, R.J. 1997) says that they can be categorized into two main types:

1. specification generation errors. This also includes reliable estimates, which is one of the 10 main factors for success (The Standish Group, International 2001). Specifying reliable estimates is very difficult, as certain estimates might not be realistic enough. IT-managers should use their collective experience and knowledge to come up with estimates that reflect the real situation;
2. unwanted/unnecessary/incorrect user functionality.

The first problem is a root cause and the second one is an impact. An impact in the problem bundle is depicted as an incoming arrow and the root cause has an outgoing arrow and sometimes also incoming arrows. In the problem bundle, you should read from left to right, where the root causes are on the left, and the impacts are to the right.

1.2.3.2 Missing requirements

During the elicitation process certain requirements might be forgotten to specify. After the system is implemented, the customer will notice that this important requirement is not reflected in the new system and requests that the new requirement will be included in the design. It is clear, the initial design is not aligned with the important requirement. Looking at requirement defects, the major problem is missing requirements. This means requirements have not been written down in the specification and were not otherwise transferred to the developers. The cause for this could be that demands were not recognized or forgotten during development. (Lauesen, S. and Vinter, O. 2001)

The causes of missing requirements are rapid changes in technology and business, and no consideration for missing requirements from multiple viewpoints (Lee, S.W. and Rine, D.C. 2004). Therefore, the specification represents an incomplete domain model and this makes manual discovery of requirements very difficult.

1.2.3.3 Contradicting requirements

Certain requirements cannot be implemented simultaneously. In such cases, a tradeoff has to be made to implement a requirement, which mains that one of these requirements has to be dropped.

All realistic computing systems have to fulfill multiple quality requirements. Though, constructing these systems is hard, as requirements tend to be conflicting. Fault-tolerance and real-time computing are generally considered conflicting, as well as reusability and performance, for example. (Bosch, J. and Molin, P. 1999)

1.2.3.4 Requirements change

In many cases where requirements change, there is no direct link to the architectural decision. The missing alignment can make it difficult to recognize whether an architectural change is needed and what has to be changed. (Hoorn, J.F. et al. 2007)

Requirements evolve as the environment in which these systems operate change (Alves, C. and Finkelstein, A. 2002). Typical changes to requirements specifications include adding or deleting requirements and fixing errors (Nuseibeh, B. and Easterbrook, S. 2000). When looking at change request in distributed environments, it is a major risk to global software development, as these changes take 2.4 times longer to resolve compared to collocated projects (Herbsleb, D. and Mockus, A. 2003).

1.2.3.5 Problems using modeling tools

Using UML modeling techniques can result in an incorrect representation of the requirements. (Simons, J.H.A. and Graham, I 1999) illustrate that there are many things that can go wrong with UML modeling and made four classifications:

1. Inconsistency, meaning that parts of UML models are in contradiction with other parts, or with commonly accepted definitions of terms;
2. Ambiguity, meaning that some UML models are under-specified, allowing developers to interpret them in more than one way;
3. Adequacy, meaning that some important analysis and design concepts could not be captured using UML notations;
4. Cognitive misdirection, meaning that the natural development path promoted by a desire to build UML models actually misleads the developer.

1.2.3.6 No basic set of requirements

There might also be too many requirements that make the alignment very difficult. According to (The Standish Group, International 2001) one of the top 10 problems for an IT-product success is the fact that there is not a basic set of requirements.

1.2.3.7 No traceability of requirements

Traceability links to the justification of the requirement and missing traceability will lead to inadequate change impact analysis and consequently poor project scope management (Damian, D. et al. 2004). Without traceability, there might be the risk of not having a clue why certain requirements are needed and this will remain unclear throughout the development life cycle. Other studies like (Blaauboer, F. et al. 2007) and (Gotel, O. and Finkelstein, A. 1994) are talking about traceability, which is different from alignment. One of the definitions that is used in these studies is defined as follows:

"Traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)."

This definition does not say how to align these requirements during all the stages in the development life cycle and how this can be measured and adjusted in a certain context.

So with alignment, certain traceability tools are needed to follow the life of the requirements. However, if requirements change, the current architecture components have to satisfy the requirements and when this is not the case, certain architectural design decisions have to be made to get proper alignment.

1.2.3.8 Incompatible requirements and design model

Another problem is that the requirements model is not compatible with a design model. A requirements model deals with the features and capabilities of the system and a design model focuses more on classes, methods and interfaces. In addition, a requirements model is made from viewpoint of the users, whereas a design model is made more from viewpoint of programmers. (Clarke, S. et al. 1999)

1.2.3.9 Insufficient quality

The problem "insufficient quality" stands for things like poor comprehensibility, poor evolveability, coupling, low reuse, high impact of requirements change and reduced concurrency in software development amongst other things (Harsu, M. 2003).

1.2.3.10 High maintenance costs

A result of not having requirements aligned with design is that the system is not providing the needed functions desired by the domain. Here, the domain consists of users and other systems communicating with the new system for example. Certain design decisions influence the maintainability of software and are of major importance, as the effort expended on changes and fixes in software is a key cost driver (Hoordijk, W. and Wieringa, R.J. 2005). There is lack of first-class representation of design decisions. Most design decisions are cross-cutting and intertwined. These problems lead to high maintenance cost. In addition, design rules and constraints are easily violated and obsolete design decisions are not removed (Bosch, J. and Molin, P. 1999).

1.2.3.11 Problems due to blended delivery model

(Damian, D. and Zowghi, D. 2003) have made a model of four problem areas during global software development. These known problems areas are: cultural diversity, inadequate communication, knowledge management and time difference. The global software development problems can be seen in figure 7.

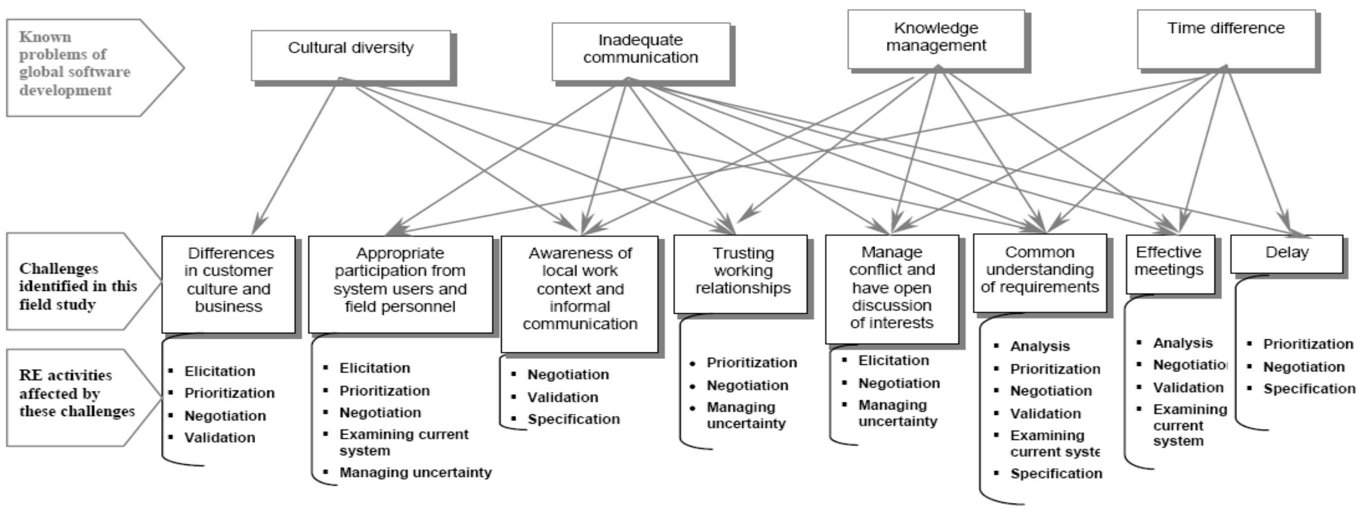


Figure 7 - Global development problems (Damian, D. and Zowghi, D. 2003)

By now, we have made a theoretical problem bundle that was validated during a meeting by a requirement analyst and a lead-architect from LogicaCMG. From the findings, we concluded that most problems are correct for the Requirement analyst and the Architect. However, the first problem bundle was missing “wrong start architecture”. The reason for adding the start architecture was, because the architecture might not be good enough when looking at adaptability, portability, usability security and other factors listed by a standards like ISO 9126 (Pfleeger, S.L. et al. 1994). This could mean that it is impossible to align functional and architectural design, due to the restrictions of the quality aspects of a system. Still, it is difficult to understand what the meaning is of alignment of requirements and architectural design. Hence, this is one of the items that is going to be researched and it will suffice to say that this issue is going to be treated in the next coming chapters.

1.3 Research Goal

The research goal of this project is:

To define the quality attributes of the requirements products in a consistent way and to reduce the design problems due to a blended delivery model

A quality attribute of the requirement product is an attribute that can improve or worsen the alignment of requirements and design. For example, a minimum set of requirement documents ensures that good design decisions can be made. Here, a minimum set of requirement documents is the quality attribute. These quality attributes are different for specific project types and require different information that can be retrieved from certain requirement artifacts.

The technical report of (Barbacci, M.R. et al. 1995) deals with identifying a system’s architecture critical quality attributes, such as availability, performance, security, interoperability, and modifiability. Indeed, this is important to specify in certain artifacts in order to be able to make good design decisions and to know why a certain design decision has been made. Hence, these attributes only concern quality attributes of the product. However, it does not include all attributes that are important for alignment of requirement and design. This research deals with quality attributes of the product, but also takes other attributes into consideration that relate to the global software development problems at LogicaCMG. Hence, these attributes have to do with four main problem categories, which are communication, culture, knowledge transfer and time, as can be seen in figure 6.

1.4 Project Context

At Result Centre, Rational Unified Process (RUP) (Kruchten, P. 2003) is mainly used for software development, as can be seen in figure 8. The RUP-model has two dimensions:

- Time is represented on the horizontal axis and it shows the lifecycle aspects of the process as it unfolds.
- The disciplines involved are represented on the vertical axis.

The emphasis for each discipline varies over time. Business modeling and requirement are mainly done in the inception phase and in later iterations you spend more time on construction and transition to implement the system. Looking at this picture of software development, there is a transition from inception to elaboration. This is where the requirements of the inception phase are transferred to the elaboration phase and are going to be used to make an architectural design. The inception phase is like a visionary milestone, where the problems, potential solutions and feasibility of the project are investigated. The elaboration phase is the core architecture milestone where requirements are prioritized and the architectural important requirements are implemented first. At the end of this phase a more reliable development plan is completed that will be used for the construction phase.

The disciplines Requirements and Analysis and Design in the inception and elaboration phase are important for my research project, as here the translation from requirements to design is being made.

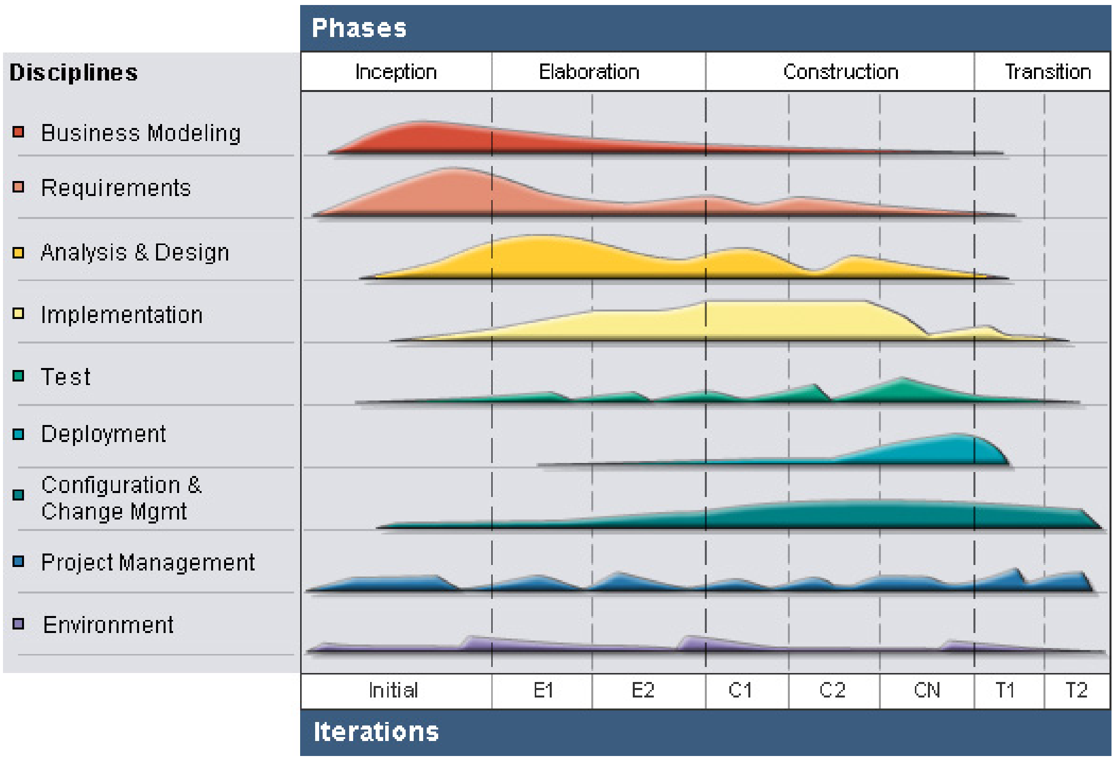


Figure 8 - Software development at LogicaCMG (IBM 2005)

Besides RUP, there are also other software development methodologies used at LogicaCMG, like Microsoft Agile for example. However, in context of this research only RUP will be investigated.

Figure 9 shows where requirements play a role in the product life cycle (Lauesen, S. 2002). In the inception step, the project is started and business goals are outlined, which results in elicitation of precise business goals and outlined requirements. After the elicitation the formulation of requirements in a finished form takes place. Before the requirements are used by developers, they must be validated. This is done in the checking step. After checking the requirements, the two parties can sign the requirements and the contract. In Tender processes the customer is going to check several proposals and there will be different proposals focusing on specific requirements of the system. When the developers are involved, requirements play an important role, as they have to be verified with the product that is being developed. Once the product is ready for deployment, it is installed, tested in many ways and verified whether the requirements are being met. After this, the requirement might have changed and this means there is need of requirements management, release planning and tracing and tool support.

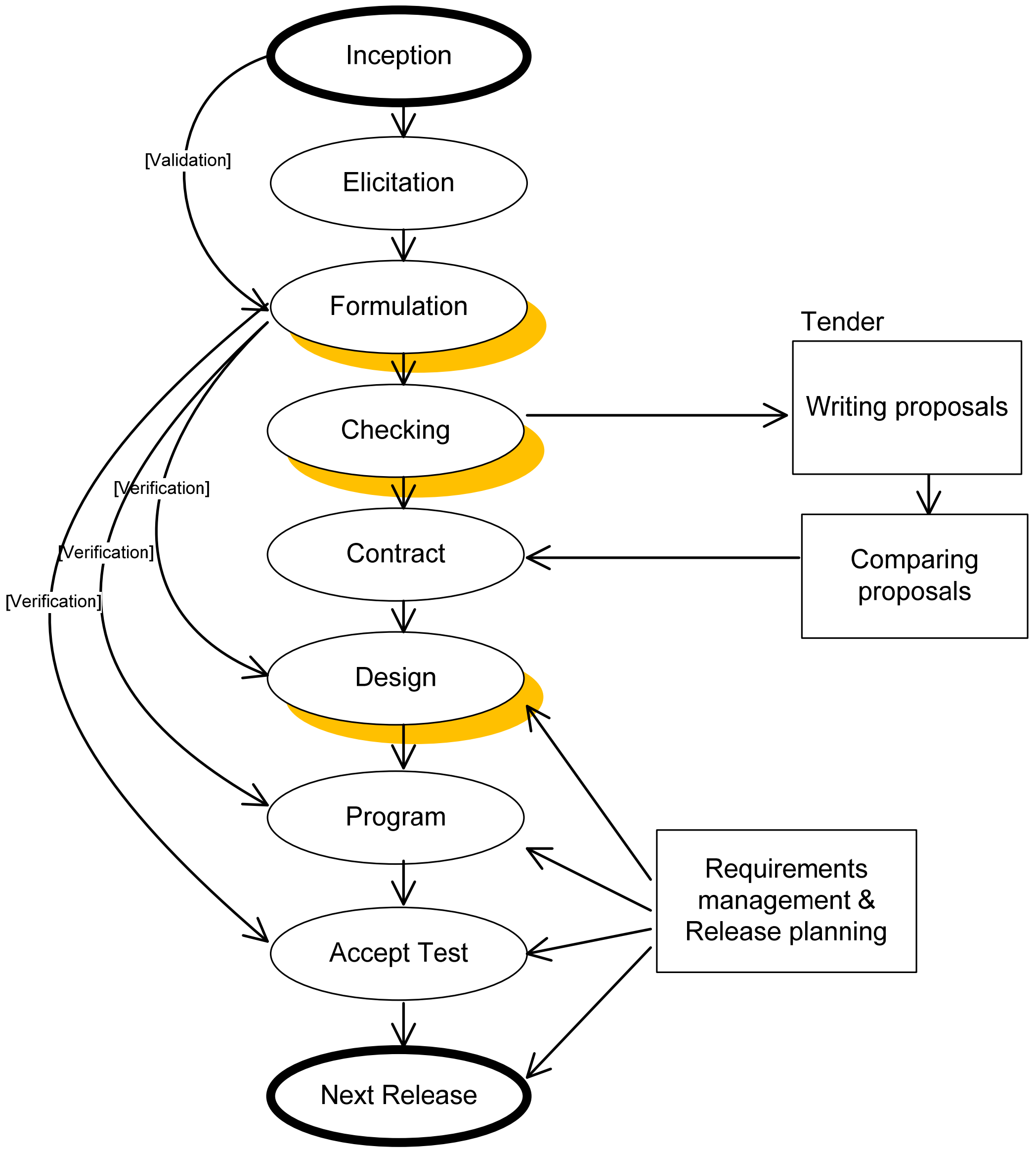


Figure 9 - Places where requirements are involved (Lauesen, S. 2002)

In figure 9, three phases are highlighted, which are formulation, checking and design. This links to the Requirements and Analysis and Design discipline in figure 8. Elicitation is out of the scope of my research project and so are the other phases that are not highlighted in figure 9.

2 RESEARCH DESIGN

2.1 INTRODUCTION

Now that the research setting, the problem statement and goals are identified, it is possible to make a proper research design for it. This chapter explains how this research tries to give answer to the research questions and elaborates the phases of the research. There are two kind of designs according to (Verschuren, P. and Doorewaard, H. 2000), which are conceptual and technical design. In this research these types of designs are also used. First, the conceptual design will be elaborated by describing the research goal, research model and research issue. The research model will be explained to give a good indication what phases exist in my research and what has to be done each phase. Subsequently, the technical design will be described by explaining how the research is going to be done. This means elaborating my approach in term of research material, research strategy and research purpose.

2.2 CONCEPTUAL DESIGN

2.2.1 Research Goal

As the research goal is already stated in 1.3, I will just repeat the goal again:

> *To define the quality attributes of the requirements products in a consistent way and to reduce the design problems due to a blended delivery model*

2.2.2 Research Model

The research model depicted in figure 10 will show the needed abstract steps to achieve the goal of my research. The advantage of such an approach is that it is fairly easy to see how the research is structured. Furthermore, it is important to have a research model to determine the theoretical backgrounds of my research.

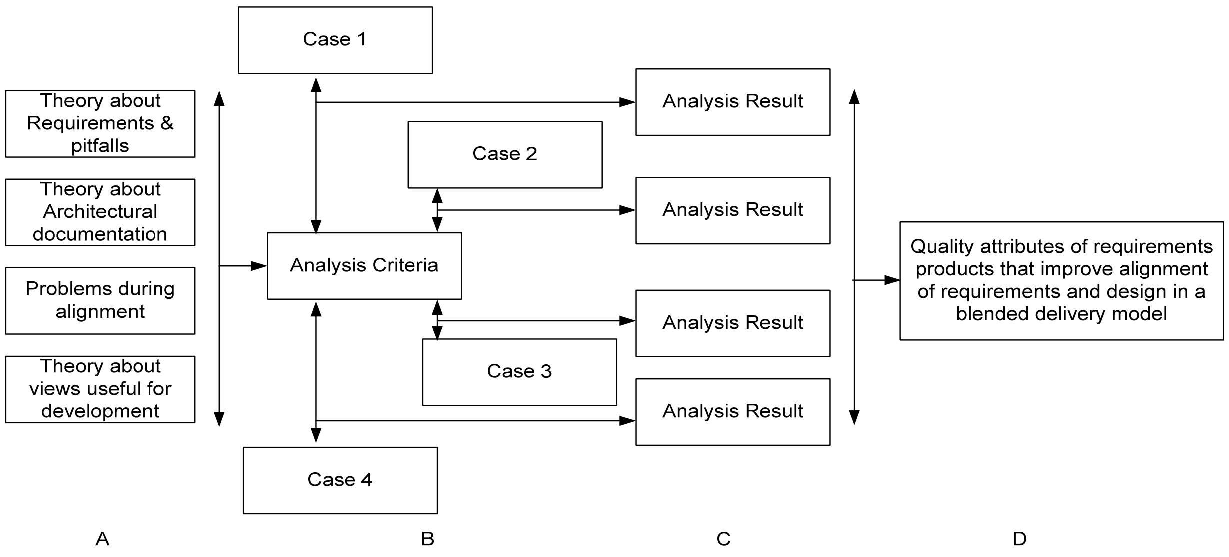

Theory about Requirements & pitfalls
Theory about Architectural documentation
Problems during alignment
Theory about views useful for development
Case 1
Case 2
Case 3
Case 4
Analysis Criteria
Analysis Result
Analysis Result
Analysis Result
Analysis Result
Quality attributes of requirements products that improve alignment of requirements and design in a blended delivery model
A
B
C
D

Figure 10 - Research model

In the first part (A) of the research model, there are different sources that will give us a better background and more information about the issues that are going to be analyzed. This will result in a conceptual framework (A) and is going to be validated by means of different case studies (B) that is also helpful for getting practical knowledge about the topic being researched. The nice thing of this approach is that there is a check whether some things might be neglected. With the knowledge obtained from phases A and B, a refined framework (C) is created that will help the Result centre of LogicaCMG to improve the alignment between analysts and designers in the blended delivery model. More specifically, quality attributes for requirements products will be constructed to satisfy the requirement demands of the designers. This will result in a better understanding about how to design a system. The refined framework (C) will be validated and recommendations (D) will be made, based on the framework made in part C. These recommendations will be the final result of my research.

2.2.2.1 Phase A

There are four different areas that are going to be investigated by means of a literature search and are the foundation for my theoretical framework. The obtained framework from this first step will be using information about requirements techniques, architectural documentation, problems during alignment and theory about views useful for development. The analyzed information will give a better understanding about alignment of the requirements specification and developers documentation. At LogicaCMG, Rational Unified Process (RUP) is used to manage projects. RUP uses the 4+1 views of Kruchten (Kruchten, P.B. 1995), each of the views addresses a specific set of concerns. Design decisions are captured in four views by architects and the fifth one is used to link and validate them. For this reason, it is important to look whether the fifth view, that is, use cases, is good enough for architects to use for their design. This is why it is important to look at different requirements specification techniques and architectural documentation, so a good overview is made about different ways to specify information about the system to be developed.

Architects and System Analysts use different information. The systems analyst focuses on system requirements and not on design issues. Architects focus more on quantified quality requirements for a system that will influence the design for example. The system analyst role is the linking pin when it comes to requirements transfer towards architects and designers.

As a final part of the theory exploration, different views for development are glanced upon to get a better understanding about the requirements specification needs of designers. A workshop was organized to define what questions are going to be asked regarding technical communication for designers. These questions are used for phase B, where in-depth case studies are done. Not only developers from the Netherland, but also people from India are asked to answer these questions. From these findings, it is possible to tell what aspects are important for alignment of requirement with design. This outcome can be used for making recommendations for the Result Centre that is using the blended delivery model. This means several important alignment concept are transformed into a framework that includes the different alignment guidelines that can be used during an outsourcing project to India.

2.2.2.2 Phase B

The next phase of the research is an empirical part to examine the framework obtained from theory and experts in the first phase. The theory might not be related to practice and there might be aspects not found in literature. The framework of phase one will be refined, based on the findings of my empirical investigation.

Four case studies will be used of previous projects and interviews will be conducted with designers and requirements specialists to test whether the theoretical framework is correct and what refinements have to be made to the alignment framework. Furthermore, solutions about tackling certain problems are obtained from the interviews and are used for the recommendations in part D. These case studies are projects of LogicaCMG with other organizations following the RUP software development method. During these projects, certain parts of the development were done in India and The Netherlands. The findings of this practical investigation will result in a refined version of the alignment framework.

2.2.2.3 Phase C & D

After this empirical part it should be clear what issues are currently present during alignment of requirements and architectural design decisions. Analysis of these theoretical and practical finding will result in several solutions to overcome the problems during alignment.

Finally, the end result will be a list of recommendations, containing the guidelines of the quality attributes important for alignment for the Result Centre of LogicaCMG. The guidelines are defined in appropriate categories that can be used for the Result Centre to optimize the translation of requirements to design.

2.2.3 Research Questions

Now that the research model is defined, it is clear what things have to be done for the research. In order to be able to reach the goal of this project, clear research questions have to be defined.

The main research question for this project is:

"What improvements can be made to the RUP development model of a multi-department software factory (Result), which results in better alignment of requirements and architectural design in a blended delivery model?"

To be able to answer this question several sub-questions have to be defined. Moreover the sub-questions can be split up in even more questions.

This research project will investigate the following research questions:

1. What are the pitfalls in the alignment of (changing) requirements and design?

    a. How to define alignment of requirements?

    b. What do we mean with requirements in this research project?

    c. What requirement techniques exist that are commonly used?

    d. What problems in alignment of requirements and design can be found in theory and practice?

    e. Which classification of problems can be made looking at theory and practice?

2. In a blended delivery environment, what are the quality attributes of the requirement products needed by designers?

   a. What are the specific alignment problems in a blended delivery model?
   b. What are the quality attributes to improve alignment of requirements and design according to literature and practice?
   c. What views for documenting requirements are most important for designers at LogicaCMG looking at different project types?

3. How can we define quality attributes for requirements products in order to achieve better alignment of requirement and design in a blended delivery model?

   a. What alignment framework can be made based on the found quality attributes?
   b. What is the internal validity of the proposed alignment framework?
   c. What are the guidelines to improve the alignment of requirements and design?
   d. What recommendations can be made after the validated solution?
   e. What issues are not tackled by using the recommendations based on the made framework?

2.3 Technical Design

2.3.1 Research Material

For research material there are five different types of sources, which are people, media, reality, literature and documents (Verschuren, P. and Doorewaard, H. 2000). In this research project only the following sources will be used:

- People: for the empirical part of my research, people are used as the main source of data. These persons will be mainly analysts and architects at LogicaCMG. Several interviews will be conducted to discover more about the issues researched.
- Literature: the first thing to look at is theory to make a theoretical framework that can be used for the empirical part of this research. The information aggregated from the scientific literature can be used for getting a better understanding of the problem domain.
- Documents: there are several whitepapers available at LogicaCMG that can be used as background information. In addition, information is available about using RUP, requirements and how the requirements are validated.

2.3.2 Research Strategy

For this research project a research strategy has to be defined that gives us a direction about how to explore the issues treated in this research. By research strategy we mean, all the related decisions about the way the research is going to be done.

There are five strategies important strategies for conducting research, which are survey, experiment, case study, funded theoretical approach and office investigation (Verschuren, P. and Doorewaard, H. 2000). In my research in depth investigation is needed and this requires case studies to get to know how requirements are used by designers in The Netherlands and abroad (Zelkowitz, M.V. and Wallace, D.R. 1998). Several projects are investigated by interviewing several people and studying several documents that gives us a better and thorough understanding about how things are done in practice. The strength of this method is that data collection of projects can occur of a short period of time. The most common way is to use qualitative methods for obtaining data. (Verschuren, P. and Doorewaard, H. 2000)

Case study research is being used due to the following reasons:

- It is important to look what, why and how alignment of requirements and architectural design happens in practice. This deals with descriptive and explanatory questions, which indicates case study research is needed.
- It illuminates a particular situation, to get an in-depth understanding of it.
- The method makes it possible to make direct observation and collect data in a natural setting.
- It enables you to investigate important topics not easily covered by other methods. (Yin, R.K. 2003)

Yin talks about *triangulation* to establish converging lines of evidence to make the findings of the specific case study as robust as possible. Besides interviewing multiple respondents during this empirical part, certain documents are used to get more information about the case study projects. These documents are available on the workspaces of the four projects that are being used for the case studies. However, not all desired documents always exist or contain sufficient information that is needed for this research. Still, if it is available, it confirms particular findings during the interview and hence, will improve the robustness of the case study results. Without doing interviews, we could never tell how to improve the RUP development model in order to positively impact the alignment of requirements and architectural design during global software development. Theory from previously published studies, whitepapers, other documents and expert opinions are used to make the conceptual alignment framework.

2.3.3 Research Purpose and Science Paradigm

Case study research can be explanatory, exploratory or descriptive (Yin, R.K. 2003). This research is exploratory, as there are few earlier studies to which references can be made for information. The aim is to look what quality attributes are important to improve alignment of requirements and design during global software development. The focus is on gaining insights and familiarity with the subject area that can be investigated more thoroughly later.

In information system research there are two distinct science paradigms, which are behavioral and design-science paradigms (March, S.T. and Smith, G. 1995). The first one uses natural science research methods and seeks to develop and justify theories that explain or predict organizational and human phenomena surrounding analysis, design, implementation, management and use of information systems. These theories try to inform researchers and practitioners about interactions among people, technology and organizations that must be managed if an information system is to improve the effectiveness and efficiency of an organization.

The design-science paradigm uses engineering and the sciences of the artificial (Simon, H.A. 1996). It is basically a problem solving paradigm and seeks to create innovations that define ideas, practices, technical capabilities and products that can be used to effectively and efficiently accomplish analysis, design, implementation, management and use of information systems. These products still have to use behavior theories, as the creation relies on fundamental existing theories that are applied, tested, modified and extended through experience, creativity, intuition and problem solving capabilities of the person who is doing the research (Markus, M.L. et al. 2002). This research makes its contribution by engaging the complementary research cycle between design-science and behavior-science to address the fundamental problems during global software development. It uses fundamental theories and extends the alignment framework by the experience of experts and the respondents of the four case studies. The design-science study guidelines, as described by (Hevner, A.R. et al. 2004) have been taken into account to properly perform this research.

3 RELEVANT ALIGNMENT CONCEPTS FROM THEORY

3.1 INTRODUCTION

In the previous chapter we have explained the research design for this thesis. The chapter started with the conceptual design, which explained the research goal, model and questions. After that, the technical design was explained, consisting of what research material is being used and defining a research strategy.

The fundamental part of my research is done in this chapter. A lot of concepts and definitions can be interpreted in different ways. In research papers, a lexical ambiguity arises when context is insufficient to find out the meaning of a single word. This chapter will define words, terms, notation and concepts within a particular context and give it a clear meaning. This chapter starts with relevant definitions used in this thesis. Furthermore, a communication model is elaborated and theory about team development is explained. Subsequently, different requirement specification styles are treated. This chapter is concluded by explaining the different architectural design views, how to capture architectural requirements, architecturally significant use cases and design patterns.

3.2 BASIC DEFINITIONS USED IN THIS THESIS

The topic of the research is about alignment of requirements and architectural design decisions. Everybody talks about alignment, but this term has different interpretations for most people. This paragraph explains definitions of the alignment framework and explains the context where this definition applies to.

3.2.1 Alignment

There are many kinds of alignment. In (Eck, P.A.T. et al. 2002), architecture alignment is from the perspective of application and the architecture. They define it as follows:

“Application architecture alignment is the process that determines the optimal fit between a software application and the existing software and business environment.”

The definition used in this research is defined in an analog way. In this research, we are talking about requirements and architectural design. The definition of alignment of requirement and design is as follows:

“In context of this thesis, requirement and architectural alignment is the process that determines the optimal fit between requirement and architectural design.”

Based on the limitations of this research we focus only on the guidelines aspects of alignment and not the actual measurement.

3.2.2 Requirement

Before we go into details of requirements, let us first define what we mean with a requirement.

According to (IEEE-Std.610.12 1990), a requirement is a:

1. *condition or capability needed by a user to solve a problem or achieve an objective,*

2. *condition or capability that must be met or fulfilled by a system to satisfy a contract, specification, standard, or other formally imposed documents,*
3. *a representation of a condition or capability as in 1 or 2.*

The definition used in this thesis is as follows:

"A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose." (Pfleeger, S.L. 1998)

There are product-level requirement and domain-level requirements. Product-level requirements specify what should come in and out of the product and domain-level requirements tell what activities go on outside the product and should be supported, in the domain. (Lauesen, S. 2002)

3.2.3 Architecture

An architecture can be interpreted as the all encompassing enterprise architecture or the more specific software architecture for example. In this thesis architecture can be seen as software architecture. The definition of software architecture that is going to be used in this research is as follows:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." (Bass, L. et al. 2003)

3.3 Communication Model

In figure 6 of chapter one different alignment problems were listed in the problem bundle. One of these problems is communication, which is an important matter for transferring requirements to designers in a blended delivery environment. Only documentation is ineffective for communication, as it does not help to resolve misunderstandings (Curtis, B. et al. 1988). There are two important aspects to communication, which are the medium and the information itself (Korn, J. 2001).

The three dimensions of communication were defined by (Morris, C. H. 1938), who has been studying the meaning and application of signs and symbols. Each dimension will be explained in the table below. During later studies other people (Shannon, C.E. and Weaver, W. 1949) (Guetzkow, H. 1965) have still used this categorization.

Table 2 - Dimensions for communication (Morris, C. H. 1938)

Dimension	Explanation
Syntax	The formal relations of signs to one another. This refers to describing the structure of common languages in terms of rules and to find general rules that can be applied to all languages.
Semantic	The relations of signs to the objects to which the signs are applicable. Semantic refers to the aspects of meaning that are expressed in a language, code, or other form of representation.
Pragmatic	The relation of signs to interpreters or users. This refers to practical consequences or real effects to be vital components of both meaning and truth.

This basic principle about medium and information was explored by (Shannon, C.E. and Weaver, W. 1949) and later by (Guetzkow, H. 1965), who made the representation that is depicted in figure 11.

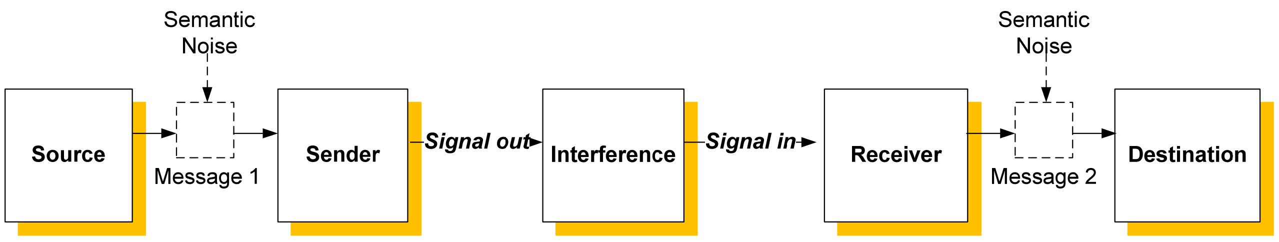


Figure 11 - The sender and receiver model (Guetzkow, H. 1965)

There are several steps in this communication model and each step is explained below:

- The source tries to encode 'message 1' as good as possible.
- The result of this encoding, 'message 1' is most of the times not good enough.
- The message is an outgoing signal, 'signal out' that is send to the receiver via a sender, channel or medium.
- During this transfer, different kind of interferences can alter the message.
- The receiver decodes 'signal in' and passes the message, 'message 2' to its destination.

Message two is interpreted according to own values and beliefs and is rarely interpreted in an objective way. Message one is only the same as message two if nothing happens during each step in this communication model. In practice this is rarely the case, moreover, during the encoding step there are certain constraints that results in a message, which does not contain all the information as we would like. In addition, the last step can contain a lot of semantic noise, which might result in a totally different message.

An example:

- Source: requirements analyst
- Sender: voice
- Signal out: only words explaining requirements
- Interference: an ambulance passing by
- Signal in: Understandable part of voicemail message
- Receiver: Hands-free-set in a car
- Destination: architect, busy driving his car

The result of this example is that message two is not the same as message one and might cause problems during systems design.

The sender always wants to achieve a goal, for us, clear requirements. The receiver also has certain goals, for example, the architect wants to make the right architectural choices. So, the content of the message is made by the sender, but the sender also has to consider the goals of the receiver. (Steehouder, M. et al. 1999)

3.3.1 Communication Noise

Unsuccessful communication can be caused by the sender, the receiver, limitations of the used medium and environmental factors. The communication can be called successful if the difference between the sender's intention and receiver's interpretation is as small as possible. During communication a lot of things can go wrong. The message can be codified in the wrong language, the message can be send to a wrong address, interference may occur during transfer and the receiver might not get the right message or understand the message. Also some sensitive issues, values and beliefs might lead to an incorrect interpretation of a message.

There are three types of interferences according to (Adler, R.B. and Rodman, G. 1988). For each of them a short explanation is given:

1. External interference; has to do with aspects outside the participants, like noise and distraction.
2. Physiological interference; are biological factors of the participants, which handicaps the participants, like speech, hearing or decease problems.
3. Psychological interference; examples are defensiveness, emotional excitements that have detrimental effects on communication

Besides these interferences, there is *semantic noise*, which is a particular diction or syntax that leads to confusion or misinterpretation of the intended message. Writing requirements can be difficult if no clear definitions are given and symbols can have different meanings. This also occurs with words or phrases that mean different things to different people.

3.3.2 Contextual Phenomena

In communication, contextual phenomena can cause problems in data aggregation (Guha, R. et al. 2004). A requirement specification also contains aggregated requirements information that has similar contextual phenomena. These contextual phenomena can be classified into a small number of categories. Per type of contextual phenomena an example is given in the table below:

Table 3 - Contextual phenomena

Class Differences. Not all entities are seen as the same class. A *person* class might be different depending on the way the class is used. Some people might say the Robot Sonny from the movie *I, Robot* is a person, because of its own will. Others might say it is a robot, as it is not made out of flesh and bone. Looking at the requirements specification, a search function might be used by one person for finding movie information on internet and by another person to only consult online help. Each person here has a different interpretation of the search function.
Propositional Attitude. This phenomenon is related to the previous one and means connecting a relational mental state to a proposition. The propositional attitude can be the simplest thoughts and expresses meaning or content and can be expressed in true or false. A person can have different mental attitudes towards a proposition like believing, desiring or hoping, which results in intentionality. Writing down a certain requirement might also be inflicted by certain propositional attitudes. Some people might believe a function is needed

to search on the internet. For the work of these persons, this function might not be really relevant. However, the employee might believe it is.
Property Type Differences. Certain property types of product might be used differently. If we express availability in a number, say 98, we do not know what this number means. Is it 98 %? 98 of what? We cannot really tell.
Point of View. This phenomenon has to do with conflicting points of view. A lot of issues are subjective. Subjective and objective data should be aggregated to make it possible to use for requirements. Subjective preferences should be made explicit, in order to know why this is important for the specific user and look how significant it is. For example, developers might think maintenance is most important whereas the end-user might think functionality is more important.
Implicit Time. Some requirements might be true at the time of writing them down, but are not maintained properly. The data does not get updated after a change. Requirements tend to change over time and should be made explicit in order to keep a correct link with the system design.
Approximations. Not all approximations reflect the real world. Daily number of visitors of a site might fluctuate regularly for example. Looking at similar sites, might give a good indication about the number of visitors. However, only a day's average is not enough. For some days traffic might be twice as high as usual. Depending on history of seasons amongst others, it might be possible to get a better value for the quantified approximation of the quality requirement in this case.

3.3.3 *Effective Communication*

Communication can be called effective when the sender and receiver achieve both their goals. Sometimes senders are intentionally sending vague messages from a strategic viewpoint. Being too specific might track a potential error to them, because they were too specific about a certain solution for example. To reach the informational goal, certain prerequisites for communication have to be fulfilled. These are:

- The information should be understandable for the receiver, without assuming certain knowledge requirements.
- The content should be structured in a decent way that makes it easy to read.
- The usage of the language of the sender should match to one of the receiver. Talking at the same level.
- The information should be represented in a nice way that motives the receiver to read everything. (Steehouder, M. et al. 1999)

3.4 TEAM DYNAMICS

Group dynamics can really influence the way team members interact and communicate with each other. The forming, storming, norming and performing model of team development was first proposed by (Tuckman, B. 1965). In order for the team to grow, to face up to challenges, to tackle problems, to find solutions, to plan work, and to deliver results all these four phases are necessary.

Next to these four phases of team development, there is one final phase where there is the break-up of the team. A team ends when their work is completed or when the organization's needs change. So the five phases are:

- Forming; pretending to get on or get along with others
- Storming; letting down the politeness barrier and trying to get down to the issues even if tempers flare up
- Norming; getting used to each other and developing trust and productivity
- Performing; working in a group to a common goal on a highly efficient and cooperative basis.
- Adjourning; this is the termination of the team. The team will share the improved process of doing development. This phase is important, as it is used to recognize the vulnerabilities of the team members.

The model of Tuckman is frequently used to describe the behavior of existing teams. Picture 12 shows the how the team development process can occur. Each team has a different way of going through the phases. The circumstances influence how the team is going to develop itself.

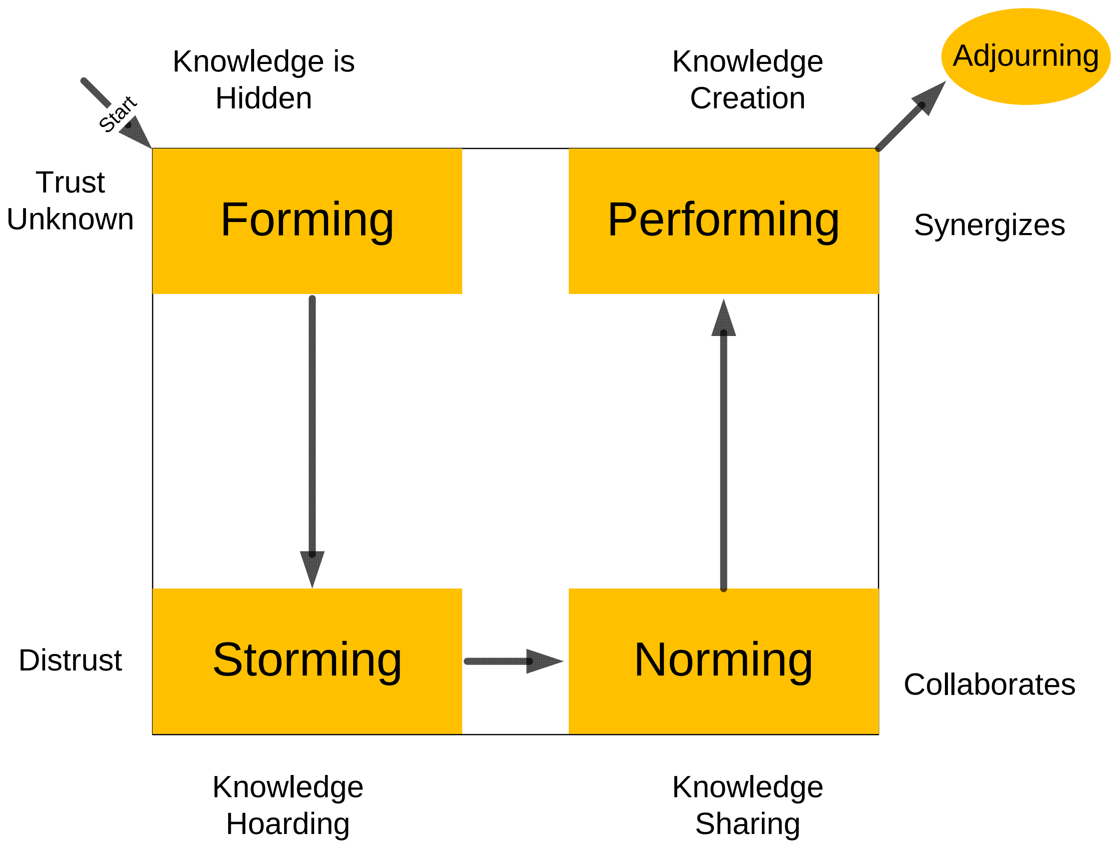


Figure 12 - The phases of team development (Tuckman, B. 1965)

At the start of forming a team not everything is working automatically and a project manager should be alert to deal with team problems. There should be time to ask critical questions and to get acquainted with each other. A good start is really important and a good way to do that is to have a kick-off meeting with the team. During this kick-off meeting different questions can be answered.

Examples are:

- What is the project all about?
- Who the people are on the team?
- With whom will you be working?
- Who is the leader?
- Are there any problems concerning the success of project?
- What are my responsibilities?

3.5 Requirements & Specification Styles

There are many techniques that can be used to specify functional requirements. Besides these functional requirements there are also quality requirements, the so called non-functional requirements (NFRs). These requirements specify how well the system must perform its functions. In many requirements specifications, functional requirements are specified thoroughly, whereas only a small part is used for quality requirements. (Lauesen, S. 2002)

Specifying functional and non-functional requirements poses a potential threat for communicating the requirements, as certain aspect might be neglected or forgotten in the specification for example. It is important to note that before system requirements are made a good problem analysis is done. Problem frames can be used for classifying, analyzing and structuring software development problems. As this should be done before specifying requirements, this is out of scope of my research. However problem framing is very useful for structuring your problems into several sub problems. Problem frames help you to define simple problem classes and makes it possible to see what concerns it raises according to the specific problem frame. The result is a problem frame that shows you what you must do to solve it. (Jackson, M. 2001)

This part of chapter three explains how quality and functional requirements can be specified. Paragraph 3.5.1 provides details about defining quality requirements and paragraph 3.5.2 explains different functional requirements specification styles.

3.5.1 Quantifying Quality Requirements

Documenting requirements is usually done using standards. Different quality standards exist for software quality. (McCall, J. A. and Matsumo, M. T. 1980) defined one of the first good standards for quality in 1980 for the US Airforce. This standard makes a distinction between three major categories:

- operation: daily use by end users;
- revision: maintenance and extension of the software and;
- transition: use of the software in new technical surroundings.

This standard of (McCall, J. A. and Matsumo, M. T. 1980) was received well by software engineers and new attempts were made to refine the standard, which resulted in the commonly used ISO 9126 standard for example. Because this standard is used by many vendors, it is going to be explained in further detail. The standard contains six overall quality factors, which can be split up in even more factors that can be measured. Defining these quality factors is important for this research, as they can be used to design a system. Picture 13 shows the detailed view of the ISO 9126 standard (ISO/IEC 9126-1 2001).

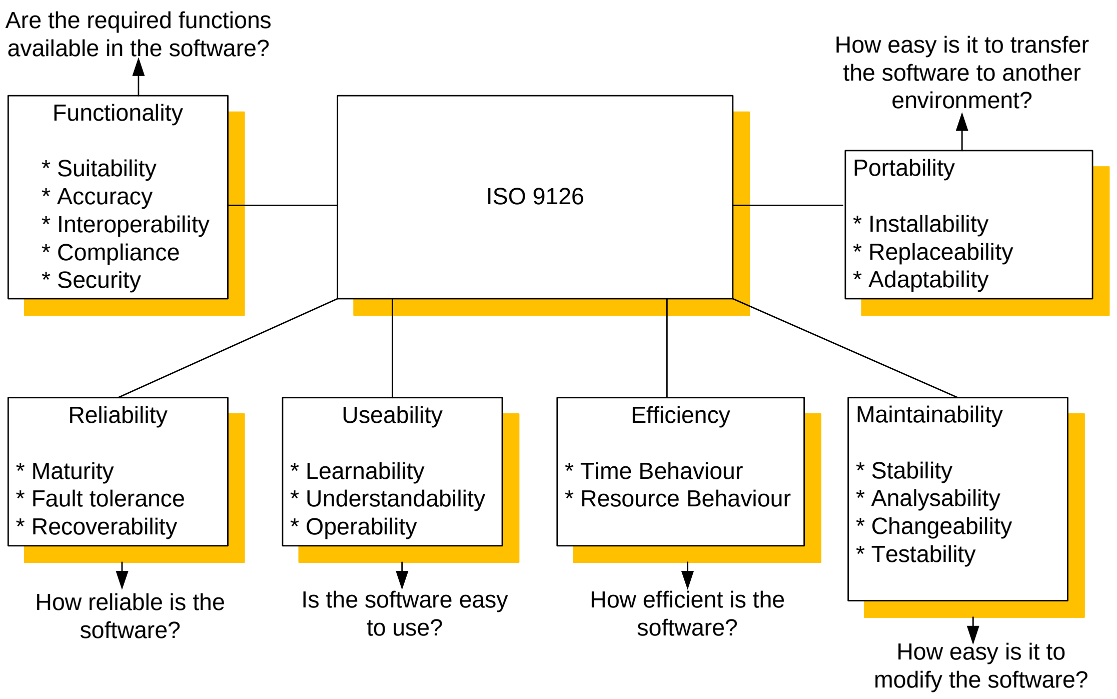


Figure 13 - Detailed view of ISO 9126 aspects (ISO/IEC 9126-1 2001)

Current specification of qualitative requirements is not done satisfactory due to the fact that there are no proper tools that can specify quality good enough (Simmons, E. 2001). Structured English is one way to specify quality requirements and is the combination of English language with the syntax and structured programming (Cin, M.D. 2000).

Another interesting concept is Planguage by (Gilb, T. 2005). Planguage is a way to specify quality using keywords. Within Planguage the quality factors are described by different attribute parameters. Table 4 shows a brief explanation of the attribute parameters.

Table 4 - The Planguage Attribute Parameters (Gilb, T. 1997)

Attribute Parameter	**Meaning**	**Used for**	**Note also**
TAG	Name of the requirement / quality attribute	Cross referencing, reuse of concepts	Avoids the need to keep repeating all the details of the full requirement
GIST	A rough informal description	Getting consensus	Useful, but rarely a clear definition
SCALE	Definition of scale of measure	Defining the concept with precision and clarity	Contractual use
METER	Definition of how we are going to measure or test the attribute in practice	Agreeing as to how the requirement fulfillment will be judged in practice	Contractual use
PAST	A known benchmark of a value in the past	Providing a baseline value	Useful reference point

MUST	A future requirement target, which is necessary for system survival	Early delivery minimum levels. Tradeoff minimum levels	Contractual minimum payment level
PLAN	A future requirement target, which is necessary for *success* and satisfaction	Understanding the full requirement. *Knowing when to stop* designing and building	Contractual *full* payment level

Each quality attribute can have several *scale, meter, past, must* and *plan* parameters. Thus, in order to describe a quality attribute adequately, it might be required to specify more than one *scale* attribute. Using Planguage can be very useful to specify the non-functionals and can be used as a work product that is used to communicate quality aspects to designers and to know the fundamental attributes parameters that were used to define the architecture. Especially in outsourcing projects this documentation might be very important to communicate, as it gives designers a better understanding about the quality constraints and makes it possible to make a good design that meets the customers' expectations. Besides the non-functional requirement, there are also architectural requirements that influence the design of a system. Capturing these architectural requirements and not only non-functional requirements is explained in paragraph 3.6.3.

3.5.2 Functional Requirements Styles

A big part of documenting requirements is done by defining functional requirements. By specifying the functional requirements, the details of functions become more explicit. This part discusses some of the many styles for specifying functional requirements (Lauesen, S. 2002). What is not treated here are functional details, as for each project there are many styles to document functions and this is different for each project. The thing that is important here is what architects use for making the design and not what details a certain function has. Per requirement style advantages are listed in terms of validation and verification. With validation the customer must be able to validate whether to see the requirement reflects the customers need. Verification is carried out to check that the product satisfies the requirements. Only paragraph 3.5.2.5 and 3.5.2.6 describe domain-level requirements, which means it describes the activities that go on outside the product, in the domain and how to support these activities.

3.5.2.1 Context Diagrams

A context diagram shows the product as black box surrounded by user groups and external systems. Arrows shows the dataflow from the system with it's surroundings. This diagram shows the scope of the product and gives an excellent view of the required product interfaces.

Advantages of context diagram:

- Validation; the context diagram gives developers an overview of the interfaces and can be used as a high-level checklist what to develop. The interfaces can be easily verified by looking at which interfaces are dealt with.
- Verification; customers can fairly easy read the context diagram, spot missing interfaces and discuss the scope of the product by discussing what is inside and outside the system.

3.5.2.2 Event List and Function Lists

An event triggers something that the system must respond to by performing a particular function. The event list shows the types of events that can occur, to which system has to respond. The event causes the system to perform a certain function and makes it possible to make a list of the functions of the system. There is a distinction in domain, also called business event and product events. Domain events arrive from to surroundings to the domain and product event arrive from the domain to the product.

Advantages:

- Validation; Customers can to some extend validate the domain level lists of events and tasks. Though, they might find it difficult to check whether all events are specified in the list. One problem is that there might be many variant for a certain event. For many variants, it is better to use task descriptions. A consistency check can be made by analysts by using a Create , Read, Update and Delete (CRUD) check.
- Verification; can be used as checklist to check which events and functions on the list is implemented.

3.5.2.3 Feature Requirements

Feature requirements are a common way to write requirements. Features are specified in plain text and many customers and analyst do not know another way to specify functionality. It is difficult to know whether users are adequately supported and business goals are covered and it can also lead to a false sense of security for user and analyst. This is a disadvantage during validation of the requirements.

Advantages

- Validation; features are the customer's language and customers love them.
- Verification; checking that all features are implemented in the final product is straightforward. For many features, this is a time consuming task.

3.5.2.4 Screens and Prototypes

The screens and prototypes style shows screen pictures and what the buttons do. Screens and prototypes might be a good idea to design final screen pictures and specify that they are requirements. This goes much further in design than feature requirements. Prototypes are not useful when the product is intended to become a commercial product (COTS).

Advantages:

- Validation; it is possible to ensure the screens are able to support the tasks and provide high usability. Still, task analysis, reviewing of screens and usability tests are needed to properly validate the specification.
- Verification; verifying that the final user interface is implemented as specified is quite simple. Experience shows, it is a good idea to repeat the usability tests with the final product. Some unwanted functionality might have been implemented during development and some issues might be forgotten.

3.5.2.5 Task Descriptions

In task descriptions, activities of the domain-level are specified and have to be supported by the requirements. No product features are described, only a description is given about what the human and computer should achieve together. Task descriptions look a lot like use case descriptions. The difference is that task descriptions describe what humans do or the computer does. Task descriptions are structured text describing user tasks. It has the possibility to specify different variants. It might be difficult for developers to translate the task description to good product functions. Task and support requirements, explained next, might help the developer with this. Non-task activities are not specified, but might require some system functionality. The table below explains an example task to check in a guest. The requirements are structured as follows;

Table 5 - Task description example (Lauesen, S. 2002)

Task:	***Checkin***
Purpose:	Provide a room to a guest. Mark it as occupied. Start account.
Trigger/precondition:	A guest arrives
Frequency:	Average 0.5 checkins /room/day
Critical:	50 guests
Subtasks:	Find room Record guest as checked in Deliver key
Variants:	1a. Guest has booked in advance 1b. No suitable room 2a. Guest recorded at booking 2b. Regular customer

Advantages:

- Validation; task descriptions speak the customers language and customers can identify special cases that should be specified. The analyst can use this as direct input by specifying sub-tasks or variants.
- Verification; it is fairly straightforward to check the final system with the task descriptions. Task descriptions provide good test cases with the many variants that can be specified. Verification is possible before the system is put into operation.

3.5.2.6 Task & Support

The previous technique did not show the existing problems and potential solutions to tackle these problems. The solution is not specified and the solving part is left to the supplier and developer. Customers might have some ideas about the solution or want to have some influence on the solution. Task and support enhance the previous technique by specifying the problems and example solutions. Task and support requirement look like task descriptions except that the subtask are divided into two columns. The left column explain the subtasks and problems in the old way of doing things. The potential solutions are explained on the right hand side. An example:

Table 6 - Task and Support (Lauesen, S. 2002)

Subtask: *Find room*	**Example solutions**
Problem: Guest wants neighboring rooms; price bargain.	System shows free rooms on floor maps. System shows bargain prices, time- and day-dependent

This approach might mean more work for the supplier, as all user's tasks have to be understood and solutions have to be specified. However this means that the supplier exactly knows the problems that have to be solved. Advantages of verification and validation are the same as task descriptions.

3.5.2.7 Use Cases

Use cases were introduced by (Jacobson, I. et al. 1994)and are good to show system interactions and to design user dialogs. Most use cases ignore the user's part of the tasks. Use cases are meant to describe a system and users and systems outside the system are not part of a use case. However, you can make extensive user profiles that describe some or all of the user activities. Use cases are not suitable for commercial of the shelf (COTS) products. We are going to use the UML definition for use cases.

"A use case is a coherent unit of functionality provided by a classifier (a system, subsystem, or class) as manifested by sequences of messages exchanged among the system and outside actors, together with actions performed by the system." (Rumbaugh, J et al. 1999)

Use cases are widely used for analysis and design and most kinds of use cases are on a too detailed level to be domain-level requirements, as they specify the design of the user interface. The perspective is from a system viewpoint and not from a user viewpoint, as no action performed by the user is specified. Figure 14 shows a use case diagram. Hence, the account system is an actor, as it transfers accounting data from the product. The text-oriented version of use case can be specified in different types, which are for example human and computer separated, computer-centric and essential use cases (Constantine, L. L. and Lockwood, L. A. D. 2001). Use cases are widely used and in some cases the customer might insist to use them. None of the use cases styles have the problem- and solution-oriented approach of Task & Support requirements.

Advantages:

- Validation & Verification; Use case diagrams can be used as top-level checklists for what to specify into more detail and what to develop. Besides the diagram, textual versions should be supplemented. Use cases are less suitable for commercial of the shelf (COTS) products, as they are very design oriented for a specific application.

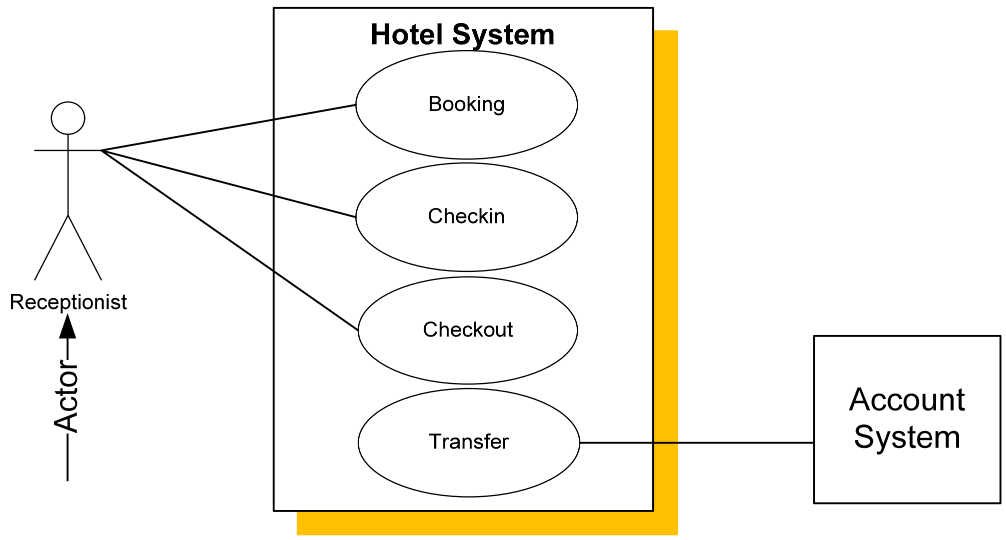


Figure 14 - Use case diagram in UML (Lauesen, S. 2002)

3.5.2.8 Dataflow Diagram

A dataflow diagram (DFD) shows activities and the data that is used and produced. It is a bubble diagram, where each bubble is an activity. Depending on the level of abstraction an activity is a process, an action, a function or a task. The bubble converts ingoing data to outgoing data. The dataflow in and out is modeled by an arrow. DFD's describe task in a technology independent way and it is a domain model, as it describes what the computer and human do together. One of the nice features of dataflow diagrams is that you can specify the exact data needs for each activity. The dataflow diagram can be specified at different levels, the second level decomposes the top-level activities into simpler activities. It is possible to decompose every activity even more; however a too detailed level might not be very useful as a requirement, since the old activities are probably going to change in the new system. DFD's do not support the problem- and solution-oriented view covered by task and support requirements. Figure 15 shows an example of a top-level DFD. DFD's are difficult to use for describing the intended functionality. It is only usable when each bubble models a specific procedure or subroutine in the code. Verification is difficult, because most of the system cannot be verified due to the fact that most functions cannot be linked to activities in the dataflow diagram.

Advantages:

- Validation; people without an IT background can quite easily understand DFD's describing domain-level aspects. Detailed level aspects might confuse the customer.

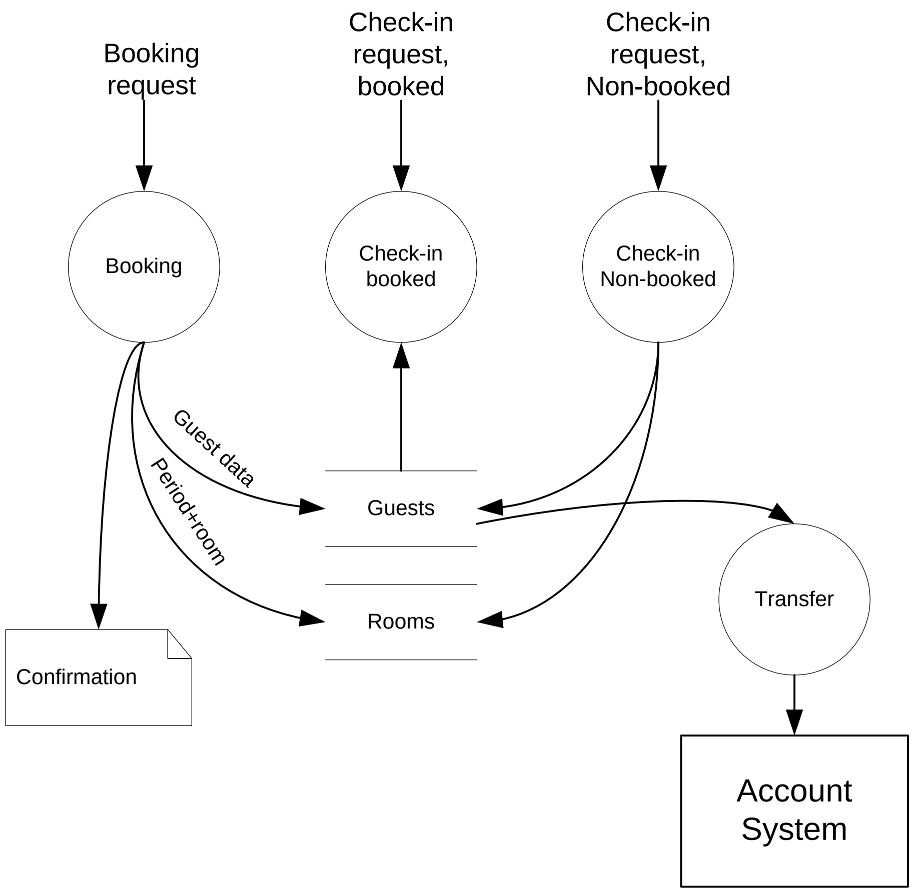


Figure 15 - DFD example (Lauesen, S. 2002)

3.5.3 Quality Criteria for Specifications

For a good requirements specification, there already is a generally accepted definition (IEEE-Std.830 1998). Below is a quick summary of the quality criteria of the IEEE 830 standard.

Table 7 - Quality Criteria for Specifications (IEEE-Std.830 1998)

Quality Criteria	**Description**
Correct	Each requirement reflects a need.
Complete	All necessary requirements included.
Unambiguous	All parties agree on meaning.
Consistent	All parts match (e.g. Context diagram and DFD).
Ranked for importance and stability	Priority and expected changes per requirement.
Modifiable	Easy to change, maintain consistency.
Verifiable	Possible to see whether requirement is met.
Traceable	To goals/purposes, in this research project to design/code.

3.6 ARCHITECTURAL DESIGN

Traditionally, software designers have been taught to use exclusively technical requirements to design a system. Modern software development looks at more facets of designing the system and tries to satisfy the needs of the stakeholder, thus reach certain goals of the stakeholder and of course, the developing organization itself.

3.6.1 Architecture Influences and Business Cycle

The architecture of a system is the result of a set of business and technical decisions. A requirements specification makes explicit some, but in most cases by far not all the desired properties of the intended system. There are different influences on the design of an architecture. The following influences can be identified (Bass, L. et al. 2003):

- Stakeholders; people interested in the construction of a software system and with different concerns and goals that might even contradict. The documents specifying the requirements influence the architecture. In paragraph 3.6.4 we can read about architectural significant use cases and how they influence the architecture.
- Developing Organization; the structure or nature of the developing organization of the system, including the assets of that organization at a certain time, like idle programmers skilled in client-server applications.
- Technical Environment; the current environment when the architecture is designed and influences that architecture, like standard industry practices or commonly used software engineering techniques.
- Architect’s Experience; the experience of the architect using a particular architectural approach. There is a good chance that the architect is going to use the same approach if he has good experience with it. On the

other hand, the architect might be reluctant to use a certain approach again if the approach was disastrous during a previous project.

The architecture business cycle that makes the architecture has many components, which are business goals, product requirements, architect's experience, architectures and system with multiple feedback loops that a business can manage(Bass, L. et al. 2003). Figure 16 shows the architecture business cycle and the architect's influences. Note that the stakeholders and developing organization itself, define the requirements qualities that the architect can use for making certain design decisions.

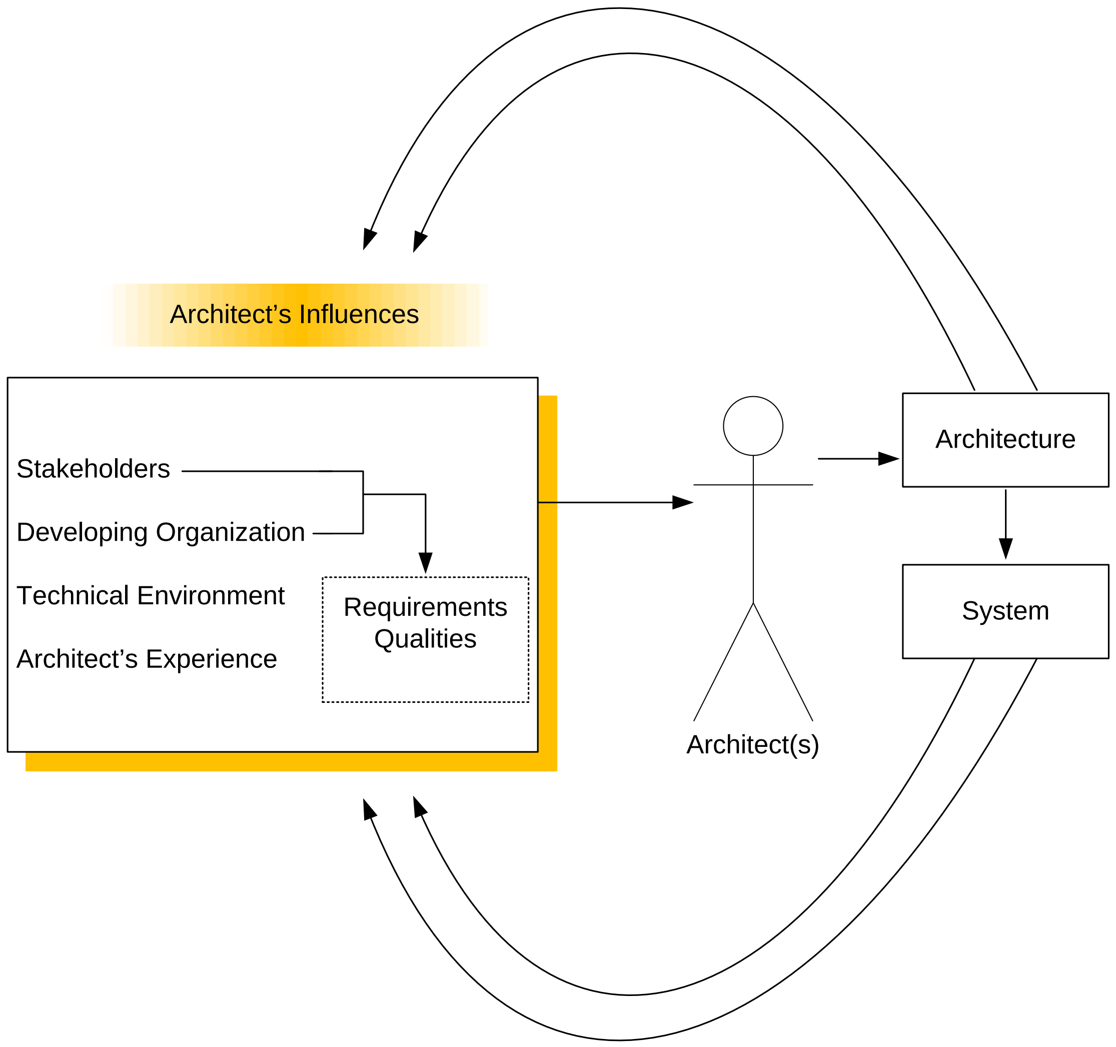


Figure 16 - The Architecture Business Cycle (Bass, L. et al. 2003)

3.6.2 Architectural Views

Different stakeholders generally have a different view of the architecture of a software system. The architect and requirement analyst looks at different aspect of the system and although the views are represented differently and focus at different aspects, they are related. A view is a representation of a coherent set of architectural elements, written and read by different system stakeholders. The structure of a system is the set of the elements itself like hardware and software and their organization. Architectural structures can be divided into three groups and differ in the elements they show. This is called the views & beyond (V&B) approach, which consist of the following structures. (Bass, L. et al. 2003)

1. Module structures; elements are modules, which are units of implementation (code based) and show the areas of functional representation. Runtime components are of less importance in this structure.
2. Component-and-Connector; the elements are runtime components (units of computation) and connection (communication among components).
3. Allocation structures; shows the relationship between software elements and the elements (non-software structures) of one or more development environments in which the software is created and executed.

Figure 17 shows a common example software architecture structures.

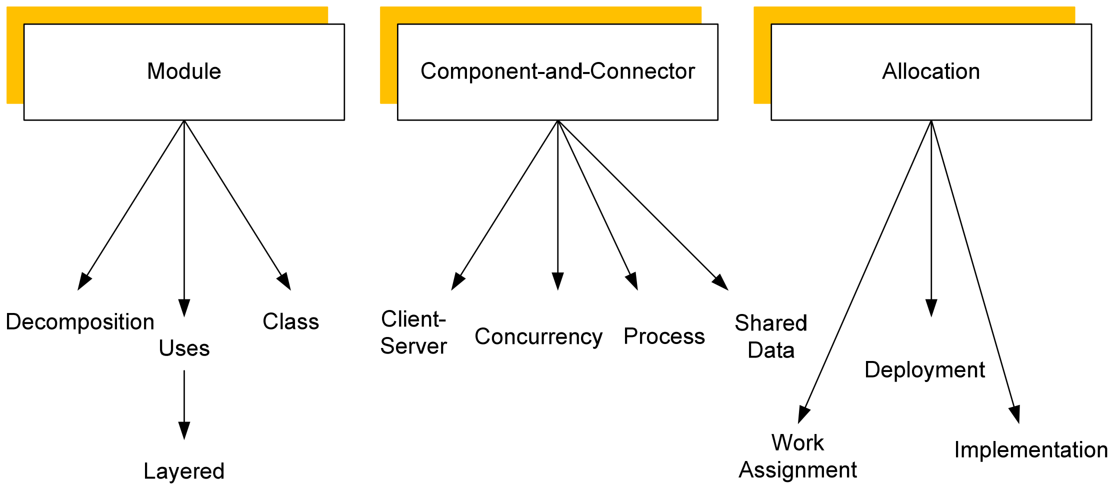


Figure 17 - Common Architecture Structures (Bass, L. et al. 2003)

There are many more structures for architectures. Below is a list of a few known models:

- Kruchten's "4+1" View Model (Kruchten, P.B. 1995)
- ISO reference model of Open Distribution Processing (RM-ODP) (Farooqui, K. et al. 1996)
- Siemens four View Model (Soni, D. et al. 1995)

One approach that is also going to be treated here is the "4+1" approach of Kruchten. This approach has become popular to use and has been institutionalized as the conceptual basis of the Rational Unified Process. The four views are needed for describing the different architecture structures and the architecture is validated by using an extra view, which are key use cases. The four views of Kruchten are:

- Logical; elements are "key abstractions", like objects or classes in the object-oriented world.
- Process; addresses concurrency and distribution of functionality and is like a component-and-connector view.
- Development; shows the organization of software modules, packages, libraries, subsystems and the units of development. This view is like an allocation view, which maps software to the external environment.

- Physical; maps other elements (processes and modules) onto hardware (processing and communication nodes).

These views are illustrated in figure 18. The scenarios validate whether the architecture is correct.

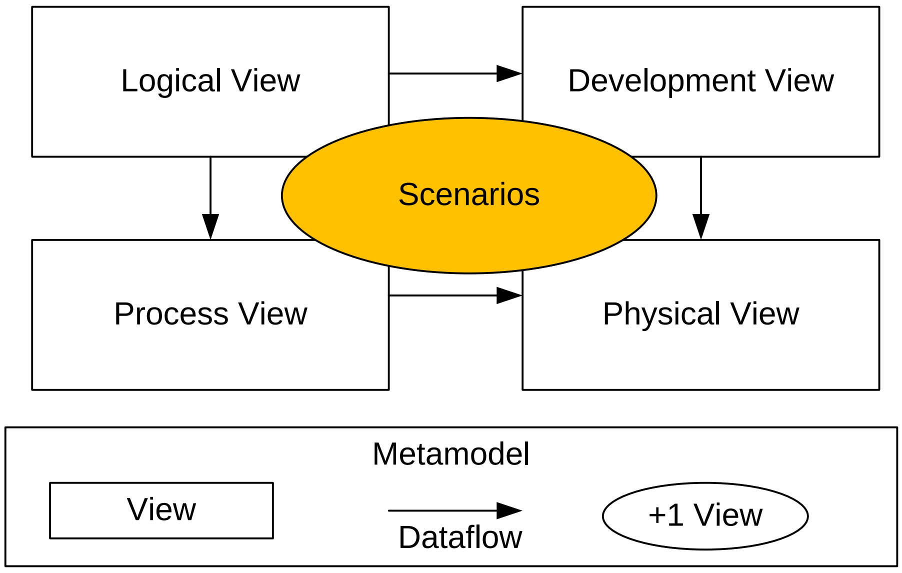


Figure 18 - Kruchten's 4+1 Views (Kruchten, P.B. 1995)

Still even with Kruchten's approach and also Siemens four model, certain aspects may be overlooked. Applying this approach means always using a fixed set of views. Recently, it is recognized that architects should produce only the views useful for the system at hand and this is done by means of the V&B approach (Bass, L. et al. 2003).

The fundamental philosophy of the V&B approach is stated as follows:

"Documenting an architecture is a matter of documenting the relevant views and then adding documentation that applies to more than one view." (Clements, P. 2005)

3.6.3 Capturing Architectural Requirements

For this research it is necessary to look how to define architectural requirements and list what information is needed to make an architectural design.

"Architectural requirements are only those requirements that are architecturally significant." (Eeles, P. 2005)

This architecturally significance can be implicit, which means the requirements have particular attributes or explicit, which means the requirements are technical in nature. Examples of implicit requirements are high-risk, high-priority or low stability. Explicit requirements are for example that the database is going to be Oracle 10g and that an online help system is compulsory. (Eeles, P. 2005)

As you can see, these requirements can be functional and non-functional. An implicit requirement is not stated explicitly in the requirements document. This is also called a tacit requirement and is defined as follows:

"When a demand is not reflected in the requirements specification, we say it is a tacit requirement. We also talk about tacit demands, i.e. demands that the user is not aware of or cannot express" (Lauesen, S. 2002)

Most requirements are implicit in practice and if you try to write down all these requirements, the result will be a huge specification that nobody can use to validate or use in development. The requirements that have to be specified are the ones that define what the developers cannot guess. *(Lauesen, S. 2002)*

A framework for classifying the architectural requirements was defined by (Grady, R. 1992). The classification system that is explained has the acronym FURPS+. It stands for:

- Functional requirements
  - Functionality
- Non-functional requirements
  - Usability
  - Reliability
  - Performance
  - Supportability

In FURPS+, the "+" refers to concerns such as:

- Design requirements; specifies or constraints the options for designing a system. An example is specifying that a relational database is required.
- Implementation requirements; states constraints of coding or construction of a system. Examples are required standards, implementation languages and time constraints.
- Interface requirements; makes explicit the interaction of an external item of a system or constraints on formats or other factors within such an interaction.
- Physical requirements; physical constraints that deal with hardware used to house the system. Examples are shape, size or weight.

Architectural mechanisms are common solutions to frequently encountered problems and are often used to fulfill architectural requirements. There are three categories of mechanism as can be seen in the table below.

Table 8 - Categories of Architectural Mechanisms (Eeles, P. 2005)

Analysis Mechanism	**Design Mechanism**	**Implementation Mechanism**
Persistence	RDBMS	Oracle Ingres
	OODBMS	ObjectStore
Communication	Message queuing	MSMQ MQSeries
	Object Request Broker	VisiBroker Orbix

Figure 19 shows the relation of requirements and mechanisms. Let us for example take persistence as the *analysis mechanism*. The *analysis mechanism* is refined into a *design mechanism*; these are RDBMS or OODBMS for the persistence *analysis mechanism.* Finally, the *implementation mechanism* is created, which is a refinement of the *design mechanism.* To stick with the persistence example, these implementation mechanisms are Oracle, Ingres or ObjectStore.

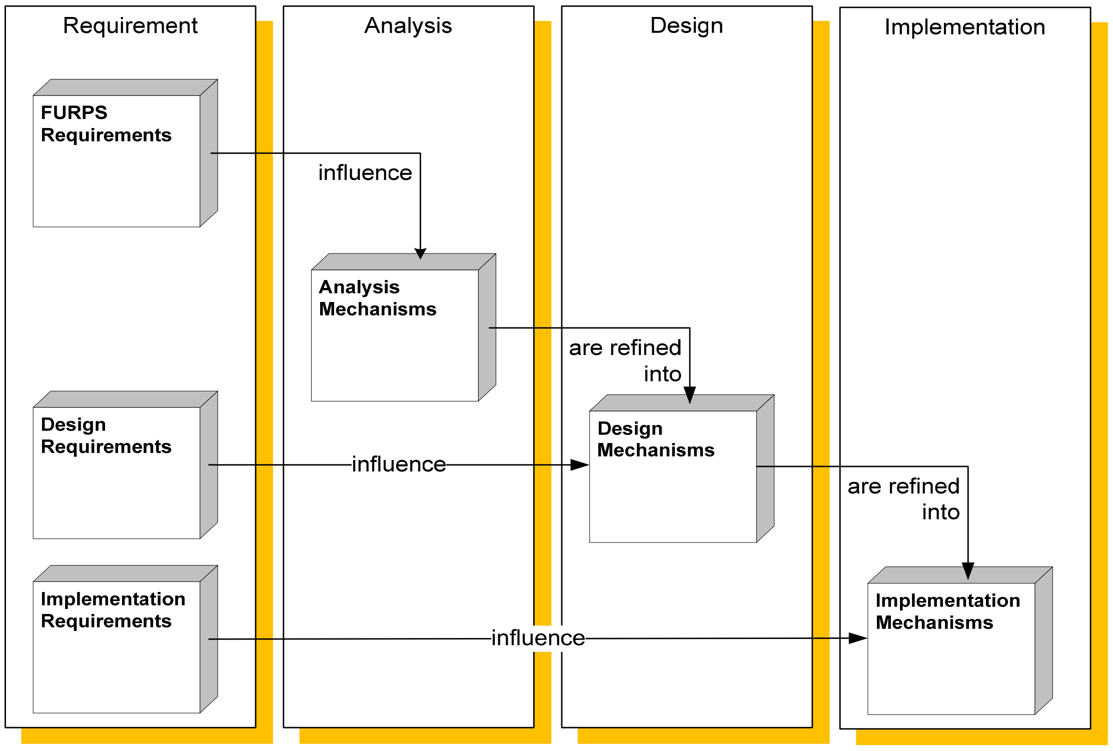


Figure 19 - Requirements and Mechanisms (Eeles, P. 2005)

In the "4+1" views of Kruchten, there are supplementary specifications that include non-functional requirements. The FURPS+ model can be used for categorizing them. Planguage (Gilb, T. 1997) can help to define the quality metrics of the system.

3.6.4 Architecturally Significant Use Cases

A critical subset of use cases can influence the architecture and are called the architecturally significant use cases (Jacobson, I. 2005). This paragraph will explain how these use cases can influence the architecture. Before knowing the use case, you must identify all use cases for the system. A distinction needs to be made between identifying and specifying use cases. Identifying has to do with scoping and exploring what the system needs to do. Specifying has to do with detailing the steps and flows in the use cases. The following steps make clear how architecturally significant use cases can influence the architecture:

1. identify the use cases in the very early phase of development;
2. prioritize and determine which use cases are most important;
3. choose use cases that influence the architecture and if there are use cases that influence the architecture in the same way, you need to just pick one;
4. explore and analyze the critical scenarios of the architecturally significant use cases that can be used to define and evaluate the architecture.

The architecture is not influenced only by architecturally significant use cases of course. It is also influenced by the platform, legacy systems that the system needs to be integrated to, frameworks, standards and policies, and so on. An overview of the influences of the architecture and the business cycle can be seen in 3.6.1.

3.6.5 Design Patterns

Design patterns can be seen as tools to improve implementing, building and maintaining a system. In general, the patterns improve your design skill as well as the cohesiveness and the quality of the product. It gives designers and builders a common implementation model that can be used by the developers familiar with these patterns. (Lasater, C.G. 2006)

If part of the design is done abroad, it is useful to know how these patterns can improve communication between designers that ultimately is going to speed up the development process during global software development. Using these patterns makes sure that developers abroad have some grip on how the system is going to be built.

Specific patterns might be used during the development of the system. There are the following pattern types:

- Fundamental patterns; deal with the fundamental aspects of object orientation.
- Creational patterns; deal with the creation of objects.
- Structural patterns; ease the design by identifying a simple way to realize relationships between entities.
- Behavioral patterns; identify common communication patterns between objects and realize these patterns. (Gamma, E. et al. 1995)

When a system is successful, more users and requirements are usually added over time. For this reason it is essential that performance can cope with the change and scale up to handle the increased load. Performance can be gained by using the combined power of different networked computers for example. Design patterns can offer a solution to cope with NFRs. (Buschman, F. et al. 2007)

Using design patterns helps during documentation and communication of the proven design solution. A design pattern does not only tell how to solve the problem, but also why the particular solution pattern is better compared to other ones and what tradeoffs have been made. NFRs can be perfectly used as description for design patterns. The only downside is that there is no systematically structured design reasoning. (Gross, D. and Yu, E. 2001) proposes a systematic treatment of NFRs in descriptions of patterns when applying these patterns during system design. Their approach structures, analyses and refines NFRs and provides guidance and reasoning when to apply design patterns that satisfy these NFRs.

3.7 SUMMARY

In this chapter we identified relevant theory that could be used for the alignment framework. The chapter started with important definitions that are used in this thesis. Subsequently, we explained a communication model, team dynamics, requirements specification styles and architectural design concepts that are used for the alignment framework described in the next chapter.

4 ASSESSMENT FRAMEWORK FOR ALIGNMENT

4.1 INTRODUCTION

The previous chapter explained relevant alignment concepts from theory that is used for my framework. Recall that these relevant alignment concepts have to do with:

- communication,
- team dynamics,
- requirements and specification styles,
- architectural design.

This chapter is going to explain the structure of the Alignment of Requirements And Design (ARAD) framework. First, the conceptual alignment framework is explained and subsequently a more detailed alignment framework is clarified. The detailed alignment framework contains quality attributes that can be used to assess the alignment of requirements and design during global software development.

4.2 CONCEPTUAL ALIGNMENT FRAMEWORK

The main aspect for my framework is communication, due to the fact that requirements have to be communicated properly in order to be able to make decisions for a specific architecture of a system. The other problems mentioned in the problem bundle depicted in figure 6 are culture, knowledge management and time problems. These problems will also be taken into account in the conceptual framework. All the issues that can occur during the transfer of requirements to architectural design can be placed beneath communication. An example is exchanging knowledge about certain design patterns. This is done by means of some kind of communication. Communication also has time constraints, as not everybody can talk with each other in geographical distributed teams. Figure 20 shows the 3 dimensions for communication together with the problems found due to offshoring development.

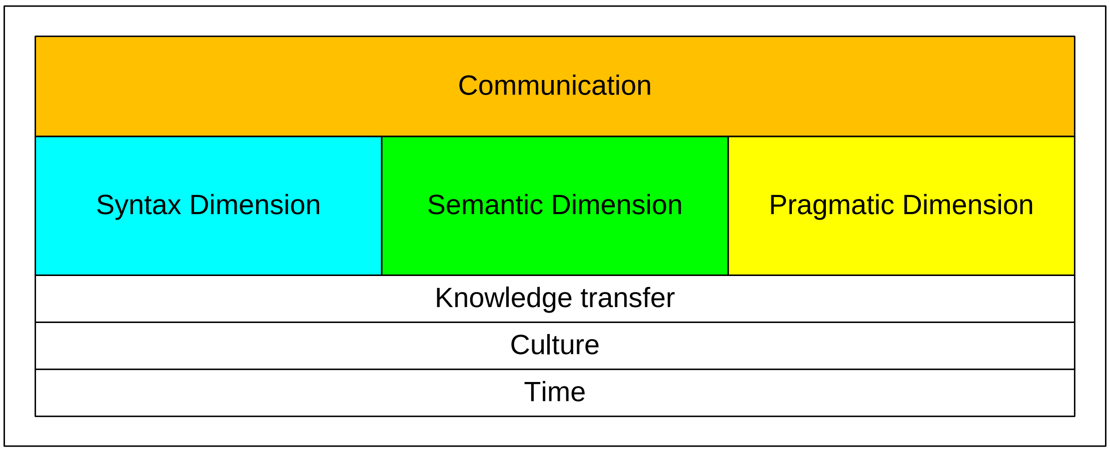


Figure 20 - The conceptual assessment framework

As you can see in figure 20, there are three dimensions for communication. These dimensions are used to analyze the alignment of requirements and architectural design. The next paragraph describes the different quality attributes for each communication dimension.

4.3 Quality Attributes of Alignment Framework

For each of these three dimensions quality attributes are specified. Recall, a quality attribute of the requirement product is an attribute that can improve or worsen the alignment of requirements and design. The framework describes what attributes have to be investigated during a project in order to assess the alignment during global software development and to be able to find solutions to improve this alignment.

For this research, it is useful to know the goal of each quality attribute in order to be able to improve the alignment of requirements and architectural design in a blended delivery model. This way, you know what you want to achieve. A good way to define goals and key questions relating to these goals is the goals question metric (GQM) approach (Basili, V.R. 1992). This approach is used, as it explains the goal at conceptual level and makes it possible to clearly define goals.

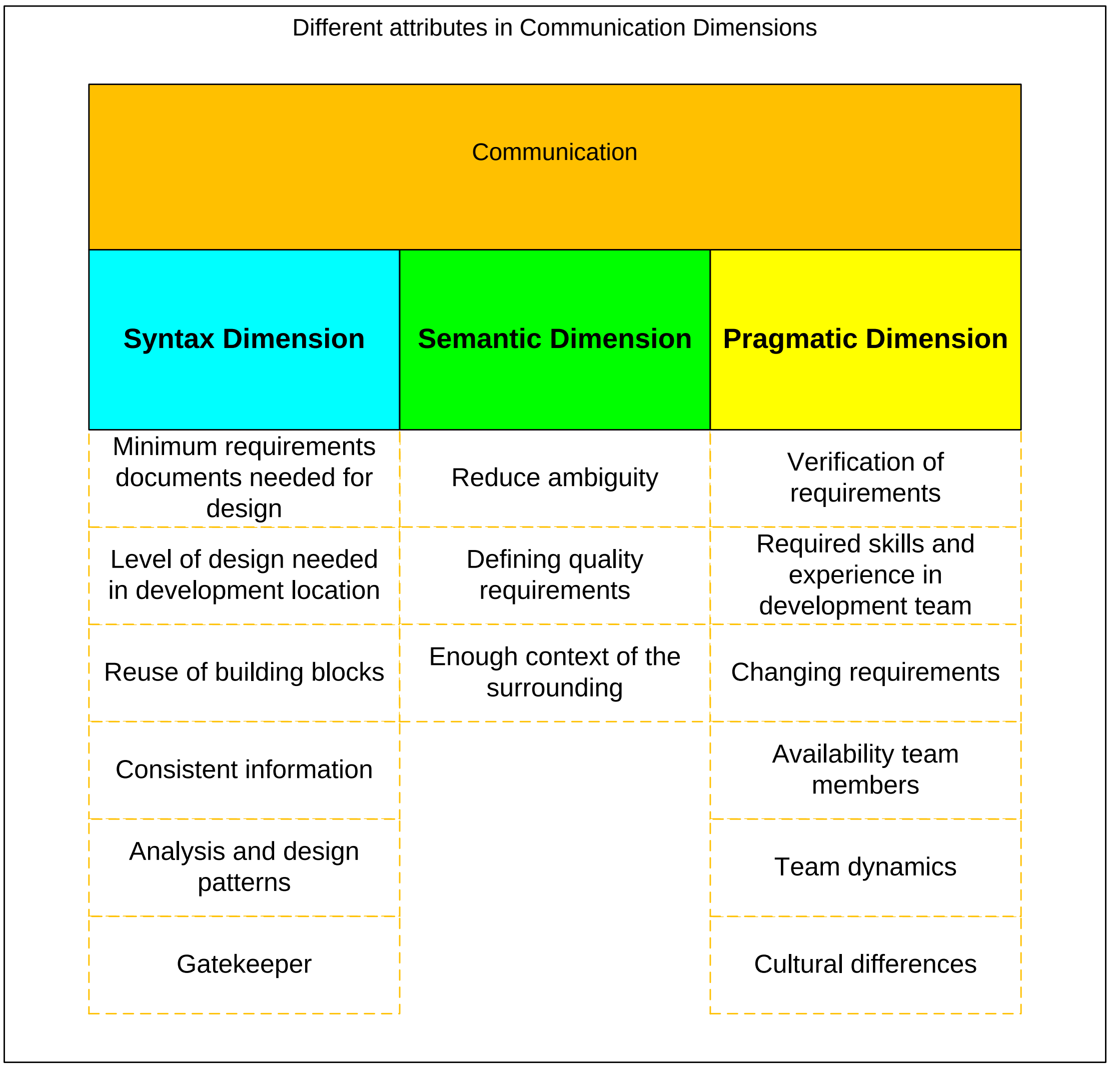
Each quality attribute in the framework has the classification as can be seen in table 9. Looking at the GQM approach, it was not possible to define metrics that improve the alignment of requirement and design during global software development. For this reason, it is not imbedded in the structure of table 9. What really matters to improve the alignment during global software development, are the guidelines found for each quality attribute. These guidelines will be consequential result from the empirical part of my research, which can be found chapter five.

Table 9 - Classification of alignment quality attributes

Description
A description of the quality attribute and why it is important.
Goal (conceptual level)
A goal has to be defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. Objects of measurement are: 1. Products: Artifacts (work products), deliverables and documents that are produced during the system life cycle. 2. Processes: Software related activities normally associated with time. 3. Resources: Items used by processes in order to produce their outputs.
Key Questions (operational level)
The key questions refers to a set of questions that is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.

Each different quality attribute that is investigated has a specific goal and key questions that can be used to assess the alignment during a project.

Figure 21 shows all the attributes of my alignment framework based on input from experts at LogicaCMG and my theoretical framework. Note that the attributes can always be placed beneath only one dimension. For example, *"Minimum requirements documents* *needed for design"* is placed under the syntax dimension, but also has pragmatic aspects. All attributes are placed beneath the dimension, which is most appropriate.


Different attributes in Communication Dimensions

Communication

Syntax Dimension	Semantic Dimension	Pragmatic Dimension
Minimum requirements documents needed for design	Reduce ambiguity	Verification of requirements
Level of design needed in development location	Defining quality requirements	Required skills and experience in development team
Reuse of building blocks	Enough context of the surrounding	Changing requirements
Consistent information		Availability team members
Analysis and design patterns		Team dynamics
Gatekeeper		Cultural differences

Figure 21 - Important quality attributes in ARAD framework

4.4 Syntax Dimension

Syntax errors are related to the structure of a certain language and can make it very difficult to understand a message. During communication of requirements to design, syntax problems can occur. This communication occurs by means of different media, like paper artifacts, online documentation or verbal communication. Important syntax attributes of the framework will be listed in this paragraph.

4.4.1 Minimum requirements documents needed for design

Description
This attribute relates to the quality criteria, *complete* of (IEEE-Std.830 1998), which is treated in chapter three. All necessary requirements should be included in the requirements specification. Requirement documents, also known as artifacts describe certain aspects of a system. A minimum set of requirement documents is needed to develop a system. However, it is not always the case that every important requirement is written down in a specific artifact. It would be useful to look at the artifacts needed to make proper design. It is really important for the sake of the success of the project to know what minimum requirements artifacts are needed. One or more of the criteria listed in appendix B should apply to the artifact and the trick is to discover the minimum level of specificity that can be used to make sure that the requirements are reflected in the design.
Goal
To define the minimum set of requirement documents needed for design
Key Questions
What artifacts were used during the specific project? Were the artifacts understandable?

4.4.2 Level of design needed in development location

Description
Outsourcing part of the development to India has several consequences concerning the level of design needed by the developers in India. Looking at figure 8, we can see the different phases of the development life cycle in RUP. It is not clear whether it is enough to have only the requirement specification to make the rest of the system. Maybe the people in India need a high level design or perhaps they really need a very detailed design. For software developers in India, for example, it might be very difficult make an architecture if it is not clear what legacy systems exist in the current architecture of a company. The level of design is really relevant looking at the blended delivery model, as developers in India for example might start developing a system without being able to consult the customer and ask questions concerning certain requirements.
Goal
To specify the level of design needed for offshoring
Key Questions
When to ship your design to the development location?

4.4.3 Reuse of building blocks

Description
Past projects might have building blocks that can be reused. It might be possible to use these specific building blocks again in another project. The reuse of building blocks is important, as it already defines knowledge and solutions that are needed during development. Not all building blocks are of the same type. (Radhakrishnan, R. 2004) makes a distinction between hard, soft and connector building blocks. Software building blocks are software entities like Enterprise Java Beans. Connector building blocks are used as the glue to connect all the components. Hard building blocks a combination of software and hardware and can be further divided into systemic and application tier building blocks.
Goal
To encourage the reuse of building blocks
Key Questions
What are the components reused for the project? How to improve the reuse of components?

4.4.4 Consistent information

Description
Information needed to make a good design, might not be consistently specified in an artifact. For example, a context diagram might list other external entities compared to the made DFD. This means certain design decisions cannot be properly made. This might result in the problem that the system build does not fulfill the needs of the customer. Consistent information refers to the consistent quality criteria for a specification as described in (IEEE-Std.830 1998).
Goal
To improve the consistency of artifacts
Key Questions
Have there been any problems in design due to inconsistent information? How was this problem tackled or how should you tackle this problem in the future?

4.4.5 Analysis and design patterns

Description
Specific design patterns might be used during the development of the system. There are several design pattern types, which are fundamental, creational, structural and behavioral patterns. These patterns help to communicate and document the design solution, which will ultimately speed up the development process (Gross, D. and Yu, E. 2001). Next to design patterns, analysis patterns can be used to efficiently transfer knowledge. The difference from design patterns is that analysis patterns reflect conceptual structures of business processes rather than actual software implementations. (Fowler, M. 1997)

Goal
To efficiently transfer knowledge using patterns
Key Questions
What patterns were used? What were the advantages of using the pattern?

4.4.6 Gatekeeper

Description
Not all information and knowledge can be managed in a proper way during a complex and large project, especially if outsourcing is being used. This means that the gatekeeper cannot decide anymore which information will go forward, and which will not. A gatekeeper in a social system decides which of a certain commodity, materials, goods, and information, may enter the system. They are also able to control the public's knowledge of the actual events by letting some stories pass through the system, but keeping others out (Lewin, K. 1947).
Goal
To improve information and knowledge exchange between team members
Key Questions
Who are the gatekeepers during the project? What information and knowledge is difficult to manage by the gatekeeper?

4.5 Semantic Dimension

Semantics is the field of using language, code or any other form of representation to specify aspects of some entity and give it a certain meaning. For system design semantics are very important, as a system might be interpreted differently by the people involved during system development. If specific aspects are misinterpreted due to improper semantic communication, this might result in a system that does not satisfy the expectations of the customer.

4.5.1 Reduce ambiguity

Description
Certain work products might be specified in very vague words and without a very clear meta-model. This means the aspect that is described might be interpreted in many different ways. Ambiguity is also one of the quality criteria for a software specification (IEEE-Std.830 1998). Ambiguity is a problem, because the different readers of the requirements specification may understand different things. The people who implement the system have a different meaning from that of the customer or users. No ambiguities also has to do with the contextual phenomena *Class Differences* and *Property Type Differences* described in (Guha, R. et al. 2004). *Class Differences*, because some things might be interpreted differently and *Property Type Differences,* as different types can be used for measuring certain aspect of a system.
Goal
To prevent ambiguous specifications in requirement documents

Key Questions
What domain knowledge and information is important to specify in requirements in order to prevent ambiguity? How to deal with ambiguity?

4.5.2 Defining quality requirements

Description
Some Approximations of the system might not be specified very accurately (Guha, R. et al. 2004). In this thesis, the term approximations has the same meaning as estimates, which is also one of the ten main factors for success(The Standish Group, International 2001). Realistic estimates of qualities like stability, fault tolerance and installability are very important aspect for an architect, as these aspects influence the architect design decisions. If we look at daily number of visitors of a site for example, this might fluctuate regularly. Looking at similar sites might give a good indication about the number of visitors. However, only a day’s average is not enough, as for some days, traffic might be twice as high as usual. By looking at the history of season amongst others, it might be possible to get a better value for the quantified approximation of the quality requirement.
Goal
To prevent wrong approximations in artifacts or in the mind of architects
Key Questions
Which artifacts did originally contain approximations errors? What estimation are difficult to make?

4.5.3 Enough context of the surrounding

Description
During system design there might be aspects of the surrounding that are important for the system designers. The surrounding consist of user groups and external systems. An example is the possibility to retrieve certain information from another system. If this aspect is not specified in an artifact and not properly communicated, it might be difficult to connect the new system to another system in a straightforward way. (Lauesen, S. 2002)
Goal
To enhance the context of the surrounding in requirement artifacts
Key Questions
Was certain environment context missing during the design of the system? What aspects about the environmental context are needed, especially when part of the design is outsourced to another development location?

4.6 Pragmatic Dimension

The practical consequences or real effects to be vital components of both meaning and truth are the area of pragmatics. In reality some things are not always the way we would like it to be. This part deals with attributes concerning the practical application of artifacts during software development.

4.6.1 Verification of requirements

Description
Verification is the process of checking whether the implemented system satisfies the requirements. As a minimum, this done during an acceptance test. Practically, this means the parties to go through the requirements one by one and check if the build product satisfies them. (Lauesen, S. 2002) Furthermore, some requirements might have changed during design. Sometimes this could mean these requirements are not updated and ultimately not implemented in the product. This aspect refers to the *verifiable* quality criteria as, explained in (IEEE-Std.830 1998).
Goal
To ensure that the product components meets the specified requirements
Key Questions
Why were certain requirements not reflected in the initial design?

4.6.2 Required skills and experience in development team

Description
One of the problems areas is the lack of experience in your development team. Certain things should be known in order to use the messages, like contract, service levels, processes and procedures, documentation and application analyses. Education institutions are currently looking how to provide global development experience, and skills to their students. (Adya, M.P. 2006) This shows that there is a need to find out what experience is relevant in practice.
Goal
To optimize the skills and experience in a development team
Key Questions
What skills and experience is relevant to have in your development team?

4.6.3 Changing requirements

Description
Some of the characteristics of a system might change over time, even in the early phases of development. The requirements written down in artifacts might be true at the time of writing them; however it might be the case some things have changed. If this is not updated in the artifacts, this might cause problems with the new system. Important system requirements should be made explicit in order to keep the requirements and design aligned. (Guha, R. et al. 2004)

Goal
To cope with changing requirements in artifacts
Key Questions
What system requirements change a lot during design? How did the development team communicate the changed requirements?

4.6.4 Availability team members

Description
Availability of team members can be a big issue in development, as it might be very difficult to contact other members using different media like phone, email, instant messaging, and so on. The difference in time zones and different holidays causes that team members living in other geographical dispersed areas have lower availability. (Guha, R. et al. 2004)
Goal
To prevent availability problems of the development team
Key Questions
Was it difficult to communicate a requirement to a person of the development team in another geographical area?

4.6.5 Team dynamics

Description
During global development team members operate in geographical dispersed areas that can also influence the dynamics of the team members. Requirement are specified by analysts in The Netherlands and developers implement the system in India for example. This might result in less dynamics within the group, as certain knowledge and information might not be communicated to other members in the team. A strong "us" and "they" feeling might eventually occur and a the group might stay in the forming phase (Tuckman, B. 1965) and will not start collaborating effectively with each other. Furthermore, it is more difficult to help out another member, as team members are not letting down the politeness barrier and do not develop trust.
Goal
To deal with group dynamics in distributed teams
Key Questions
Did the development in geographical dispersed areas result in less dynamics of the team and why did that happen?

4.6.6 Cultural differences

Description

During a project, different cultures exist even if the project is only done within a single country. For outsourcing, this gap usually is even bigger and there might be problems about how to communicate certain aspects appropriately. It is not clear how this difference in culture is causing problems during communication of the requirements to design. Measurement of different cultures can be done by the five dimensions defined by (Hofstede, G. 1994). These dimensions are:

- Power distance (PD); the extent to which the less powerful members of organizations accept and expect that power is distributed unequally.
- Individualism (I); the extent to which people are expected to stand up for themselves, or alternatively act predominantly as a member of the group or organization.
- Masculinity (M); refers to the distribution of roles between the genders which is another fundamental issue for any society to which a range of solutions are found. Masculine cultures value competitiveness, assertiveness, ambition, and the accumulation of wealth and material possessions, whereas feminine cultures place more value on relationships and quality of life.
- Uncertainty avoidance (UA); reflects how a society attempts to cope with anxiety by minimizing uncertainty.
- Long-term orientation (LO); the importance attached to the future versus the past and present. In long term oriented societies, thrift and perseverance are valued more; in short term oriented societies, respect for tradition and reciprocation of gifts and favours are valued more.

Goal

To cope with cultural differences in the development team

Key Questions

What were the cultural dimensions of the project team in The Netherlands and in India?

What pragmatic aspects in communication of requirements to design went wrong due to cultural differences?

4.7 REFINEMENTS OF FRAMEWORK

The ARAD framework has been altered several times after feedback from the interviewees, experts and my supervisors. This part shortly indicates what has been changed in the alignment framework.

4.7.1 Altered attributes

- The quality attribute "*Design patterns*" was changed, as not only design patterns are important, but also analysis patterns. The resulting quality attribute was named *"Analysis and design patterns".* Analysis patterns can improve the alignment of requirements and design, as is gives developer a better understanding about the customers' processes.
- The quality attribute *"Time"* was altered into *"Availability team members".* This includes not only the time difference, but also holidays and the like.
- *"Required skills and knowledge to use artifacts",* was changed into *"Required skills and experience in development team"*, as it is more useful to look what skills and experience is needed in a development team and not to only focus on the artifacts.

4.7.2 Added attributes

- The alignment attribute *"Team dynamics"* was found to be important by experts and my supervisors, as a distributed team probably has more problems to communicate requirements.

4.7.3 Deleted attributes

- *"Reviewing artifact"* attribute was deleted, as this is a means and not a quality attribute for alignment of requirements and design.
- *"Different point of view"* attribute was deleted. This is not an alignment attribute, but rather something that is reflected in the type of requirements artifacts used. Hence, this is the *"Minimum requirements documents needed for design"* quality attribute of the alignment framework.

4.8 SUMMARY

This chapter explained the alignment framework that can be used to assess whether the requirements are aligned with the design during global software development. Several quality attributes were identified from theory that correspondent to the syntax, semantic and pragmatic dimensions during communication. Moreover, the important refinements of the ARAD framework have been quickly described at the end of this chapter.

5 VALIDATION OF ALIGNMENT FRAMEWORK

5.1 INTRODUCTION

The previous chapter contains the ARAD framework that is ultimately going to be used to know how to improve alignment of requirements and architectural design. The attributes most important for alignment were identified here. This chapter is going to use in-depth case studies to validate whether these important quality attributes can also be found in practice at LogicaCMG and identify any missing elements.

The first part will explain the limitations of the cases studies used for my research and the rationale. Furthermore, an explanation will be given which case studies were selected, including the context of the different case studies and selected respondents. Subsequently the interview results will be presented. For each case, the attributes of the framework will be analyzed based on the findings of the interview results.

The next chapter will contain the specific solutions relating to the different quality attributes of the alignment framework.

5.2 LIMITATIONS OF CASE STUDIES

In the technical design part of chapter two, people were selected as main source of data for the case studies. The selected people were asked a number of questions concerning the alignment attributes of my framework during the interviews. There might me limitations looking at the availability of the respondents that are used for each case study, as not everybody might still be working at LogicaCMG or might not have time for an interview.

Next to interviews for different case studies, there are five other common sources of evidence in doing case studies. These are documentation, archival records, direct observations, participant observations and physical artifacts (Yin, R.K. 2003). Documentation is also used as knowledge source in order to make the results more robust. It is a danger that not everything of the framework can be investigated, as documentation might not be always available for all the projects and interviewees might not know everything related to the asked questions.

It is important to keep in mind that the people interviewed answer the question relating to a specific alignment attribute and this might not reflect the real situation. The interviewee might have experience doing things one specific way and state that this is only important. However, another person might say totally different things. This means there is risk of bias during the case study interviews.

Another limitation is perhaps the structured way the interviews were conducted. The next paragraph explains more about this. Still, there was space for more in depth investigation. Almost all relevant findings from the conducted interviews could be associated to a quality attribute in the alignment framework.

5.3 DATA COLLECTION APPROACH

There are several steps that should be prepared before collecting the data during the case studies. The steps taken in this research are to properly prepare before doing an interview, develop a case study protocol, and conduct a pilot case study interview.

Conducting interviews can be performed in two different ways according to Yin. The first type is the open ended interview, where the interviewee expresses his opinions as well as facts in order to get new insights through an unstructured interview. The other type is the focused interview, which is a list of topics that have to be discussed. The focused interview is often performed by making use of a case study protocol to structure the interview. (Yin, R.K. 2003)

In this research, focused interviews are used that are based on a structured interview protocol, which can be found in appendix D. This makes sure all quality attributes defined in the alignment framework are treated during an interview. The focus during the interview was on the selected case study, though general findings were also treated for each quality attribute of the alignment framework. After the interview was processed in a document, the respondents were asked for feedback to check if everything was correctly stated.

Yin states a researcher needs the following skills when performing an interview: the ability to ask good questions and to interpret the responses, be a good listener, be adaptive and flexible to react to various situations, have enough knowledge about the issues treated, and be neutral about the opinions of the respondents. Before the people were interviewed, the interviewer was of course prepared thoroughly about the different quality attributes, which ensured that additional explanation could be given, if this was desired. Before the interview was conducted, the respondent already got a document explaining the background information related to my alignment framework. This ensured that definitions and the context were clear before the interview was performed.

The results of the interviews were dealt anonymously in order to improve the quality of the interview results. For confidentially purposes only a description of the project is given and no names of the persons and the organizations involved can be found in this thesis. Particular topics were kind of difficult to ask, as this requires the respondent to have comments on other team members or admit mistakes the persons had made themselves for example. For example, to admit that there was not enough knowledge about the RUP methodology in a team is an issue that someone might be afraid of to admit, when this is not done anonymously. This way, there was no reason for the interviewee to be afraid of their personal actions and that they are held responsible for a certain mistake.

To test the questions defined in the interview protocol, a pilot case study was performed. This made sure that any problems regarding missing or wrong alignment attributes could be dealt with. If there were problems regarding to data collection, there was still the possibility to adapt it.

5.4 DATA COLLECTION EXPERIENCES

Before data was gathered, the questions asked per alignment attribute were validated by my supervisors and experts at LogicaCMG. As recommended by (Yin, R.K. 2003), a pilot interview was performed. An available analyst from the first case study was interviewed about the different quality attributes to check whether the interview protocol and alignment framework needed to be changed. The results from this interview were quite promising and enough information could be gathered from the chosen data collection approach. Still not all attributes were correct or already identified, as turned out later. The framework has been refined after obtaining more information from multiple respondents.

The strategy in the beginning of my research was to do at least two or three interviews for each case study. Furthermore, not only Dutch respondents were selected, but also Indian. The total number of respondents used for all the case studies is ten, of which only two are Indian. Before the case study part of this research, it was not clear that a lot of the design is still done in the Netherlands and that it was quite difficult to arrange interviews with Indian developers or project managers.

The Indian respondents did not understand the questions about team dynamics and cultural differences, as they could not answer them. Besides the RUP-based projects, an additional interview with an Indian Lead developer was conducted about an agile project to gain additional information about their point of view. From all the interviews done with Indian respondents, one thing I noticed was that they were very positive about most things. On the other hand, Dutch respondents were more critical and gave me better a better understanding about the problems related to alignment of requirements and design during global software development.

For the first, second and third case study, enough information was gathered from different respondents. However, for the last case study only one person was interviewed. This was because the architect and the Indian developers were not working for LogicaCMG anymore and thus could not be approached. The system analyst was not available either, as he was busy with three other projects and could not spare any time for an interview.

It was difficult to get the respondents to fill in the importance of the quality attributes, as most of them were unavailable due to summer holidays or busy with important projects. The result was that only ten respondents have filled in the sheet to indicate which attributes are more important or not. So the result of this has perhaps limited validity due to a small data set.

5.5 Project Description

Each software project is different and the involved product has different attributes that can be measured. Requirements of course, are not always the same for different project types. Next to these project types, there are some characteristics for measuring the complexity of a project. For each project a description will be given. Other indicators used for project context are listed in the table below. Appendix C shows the project attributes for each case study.

Table 10 - Project characteristics

#	Explanation
1	**Size of development team** The number of people involved during development of the product.
2	**Time estimated and time needed in reality** The time needed to design, construct and test the system compared to the estimated time to design, construct and test the system.
3	**Development language** The language that was used to code the system. Examples are C++, C# or JAVA.
4	**Size of the System** Function points (FP) is used to calculate size of software. The amount of functionality that is relevant to and recognized by the user in the business is counted, which gives the number of function points. This FP approach tries to solve the variability problem of the source lines of code approach (SLOC), as the number of lines of code might vary with different implementation languages for example. (Pfleeger, S.L. 1998)

5.6 CASE STUDIES SELECTED

The case studies selected are projects based on the RUP software development method, as this is the main development method at the Result Center. Due to constraints of my research the focus is only on the RUP development method. For confidentially purposes only a description of the project is given and no names of the persons and the organizations involved can be found in this thesis.

For each different case specific project characteristics were asked during the interview. This part will explain the context of every different case. Four projects are explored in depth by interviewing different people involved during system development. All details concerning project characteristics can be found in Appendix C.

5.6.1 Case "Alfa" - Service for health insurances

The service for health insurances was a project to make a system that supports civil servants with their health insurances. The current system that is used for the primary process is not prepared for the new health insurance law.

The health service wants to be ready for the future. This means the system:

- is using the same kind of functions of the old system when dealing with health insurances. Using old way of primary process should still be possible.
- has the functions needed to adjust the health insurances to make them compliant with the new regulations. So this is the new way of supporting primary process.
- has the possibility to offer different health insurances to multiple parties.

Table 11 - Project characteristics Case Alfa

#	**Explanation**
1	**Size of development team** 40 people, 20 in The Netherlands, 20 in India
2	**Size of the system:** Large

The background of the different respondents is shown in a separate table for each case study. It will be clear what their role was during the project and what educational background they have. Furthermore, the number of projects they were involved with can be found in the table below.

Table 12 - Respondents for Case Alfa

Respondent	**A**	**B**	**C**
Abstract role	Analyst	Developer	Manager
Role	Requirements specifier	Lead architect	Project Manager India
Background education	Technological Innovation Sciences	HEAO & Business informatics	MSc. Civil
Number of projects involved	15	10+	10

5.6.2 Case “Bravo” - Service to support primary work processes

The service supports primary work processes of part of a Dutch government ministry. It concerns amongst other things planning, monitoring and dealing with inspection businesses and building and maintaining the electronic business file. There are three main user categories based on different executive boards. Since the 90’s LogicaCMG has been involved in supporting the different executive boards and the expertise gained in this area has been used to replace the old system with this new one. For each different executive boards there is a subproject. So this project actually consists of three subprojects. The respondents for this case study were involved in mainly the first and third subproject.

For the design and development RUP has been used in combination with UML modeling. The analysis part has been done in The Netherlands and building and testing has been done in India. For the development of this application the following tools were used:

- Microsoft C# .NET & Microsoft SQL Server 2000
- Catalyze
- IBM Rational Suite (ClearQuest, ClearCase)

The respondents for this case study can be seen in the table 14.

Table 13 - Project characteristics Case Bravo

#	Explanation
1	**Size of development team** For first subproject: 24 people (14 in The Netherlands, 10 in India) For third subproject: 22 people (8 in The Netherlands 14 in India)
2	**Size of the system** Large

Table 14 - Respondents for Case Bravo

Respondent	**A**	**B**	**C**	**D**
Abstract role	Manager	Analyst	Manager	Manager
Role	Project manager 1st subproject	Information Analyst 1st subproject	Project Manager 3rd subproject	Project manager India 3rd subproject
Background education	HTS – Electronics & Computer Science	Technological Innovation Sciences	Aerospace Engineering Delft	MSc. Software Systems
Number of projects involved	15	12	10+	10

5.6.3 Case "Charlie" - Payment system for high priority transactions

A new electronical payment system, intended for high priority transactions was made for two banks that merged, as they wanted to have one system. The system can be categorized as a highly event driven system containing workflow management and special authorization elements. This project was a high profile project, because the higher management was looking at it from the bank, as well as LogicaCMG itself. The old interfaced applications that communicated with another banking system needed to be altered, as a new system is being used to do transactions between banks in the Netherlands. The old system makes use of Interpay services, but due to a decision of the bank at European level, Target/2 EBA services have to be used. The new application will be used for two separate banks that have merged and will be using only one system for high priority transactions.

There are not a lot of high priority transactions; however each transaction can contain a large amount of money. There was special attention for security, efficiency and reliability during the project.

The old system was using a Windows platform. For continuity purposes a standard production platform was chosen for the new system, which turned out to be IBM Websphere and Advanced Interactive eXecutive (AIX). AIX is a proprietary operating system developed by IBM based on the UNIX System. IBM websphere is a platform for governing software and systems delivery and is designed to set up, operate and integrate e-business applications across multiple computing platforms using web technologies. The respondents for this case study can be seen in the table 16.

Table 15 - Project characteristics Case Charlie

#	**Explanation**
1	**Size of development team** 10 people (5 in The Netherlands, 5 in India)
2	**Size of the system** Small

Table 16 - Respondents for Case Charlie

Respondent	**A**	**B**
Abstract role	Developer	Analyst
Role	Lead Architect	System Analyst
Background education	Technical Computer Science, Open University Air & Space Aeronautics Architect at Fokker	HEAO & business informatics
Number of projects involved	25 (5 at LogicaCMG)	10

5.6.4 Case "Delta" - Medical system for psychiatry specialists

The medical system for psychiatry specialists was intended to be used for automation of patients' records. The system can be used to maintain a patient file and make exchange of patients' files easier. It does not indicate in which order a psychiatric specialist does his tasks, so it does not follow one specific process. It does contain however all task a specialist can do. The system uses many pull down menus that makes it possible to view and change information of patient records. The product runs on IBM Websphere, which is a platform for governing software and systems delivery. The respondents for this case can be found in table 18.

Table 17 - Project characteristics Case Delta

#	Explanation
1	**Size of development team** 25 people (10 in The Netherland, 15 in India)
2	**Size of the system** Medium

Table 18 - Respondents for Case Delta

Respondent	**A**
Abstract role	Manager
Role	Project Manager
Background education	Computer Science, Haagse Hogeschool
Number of projects involved	10+

5.7 Analysis Interview Results

Each case study is explored in depth by means of several interviews. Appendix E shows the different interviews that are conducted for all the cases. For all quality attributes of the classification framework, the results of the interviews are presented.

The results are structured in the syntax, semantic and pragmatic dimensions of my alignment framework. Table 19 shows the legend, where each color correspondent to a communication dimension.

Table 19 - Legend of communication dimensions

Dimension	Color
Syntax	🟦
Semantic	🟩
Pragmatic	🟨

5.7.1 Overview quick assessment

The respondents were asked to fill in a quick assessment of the project. General things were asked like outcome of the project, but also whether the right artifacts have been used and so on. Table 20 shows the results for the different case studies. The left column shows what aspects have been asked to get a quick impression about the project. As you can see the quality of the artifacts was quite bad. Furthermore there was no proper use of requirements management tools like RequisitePro for case studies Alfa and Bravo.

What was striking to see at case study *"Alfa"* was that respondent B, the Lead Architect indicated that the architectural significant requirements were not clearly defined. However, the Analyst and Project Manager thought this was good. This shows that the architect was not able to make a good design decision, based on the architectural significant requirements. Another interesting finding can be seen at case study *"Bravo"*. Respondent C, the Project Manager in The Netherlands indicated that not all right artifacts were used, which is in contrast with respondent D, the Project Manager in India.

Table 20 - Quick assessment Case Studies

Case Study	**Alfa**			**Bravo**				**Charlie**		**Delta**
Aspect \ Respondent	**A**	**B**	**C**	**A**	**B**	**C**	**D**	**A**	**B**	**A**
Outcome of the project	Fair	Fair	Good	Good	Fair	Fair	Good	Fair	Good	Good
Use of right artifacts	Good	Fair	Good	Fair	Good	Poor	Good	Good	Fair	Good
Product satisfies all defined requirements	Good	Fair	Good	Good	Good	Fair	Good	Good	Fair	Good
Architectural significant requirements were clearly defined	Good	Poor	Good	Good	Fair	Fair	Good	Good	Good	Good
Knowledge of development team	Fair	Good	Good	Fair	Good	Fair	Good	Fair	Good	Fair
Sufficient use of requirement tools	Fair	Fair	Fair	Fair	Good	Fair	Fair	Good	Good	Good
Artifacts of sufficient quality	Fair	Fair	Fair	Fair	Fair	Poor	Fair	Good	Fair	Fair

5.7.2 Results Case Study "Alfa"

Attribute	Findings
Minimum requirements documents needed for design	• Needed artifacts in RUP: Vision, Storyboard, Use Case Model, Use Case Descriptions, Supplementary Specifications, Glossary • Mapping of screens & data missing
Level of design needed in development location	• Artifacts needed of analysis and design: SAD, Data Model • Detailed design done in The Netherlands in practice • Enough contextual information and data model after analysis phase makes technical design possible in India in theory • Contextual information is needed for non-standard projects
Reuse of building blocks	• Microsoft framework was used and is well known
Consistent information	• User interface information not consistent • Function not consistently specified in Use Case • Requirement specifier documented unrealistic requirements • Document exactly how UI should look like • Review UC with at least Tester, Builder and customer
Analysis and design patterns	• Microsoft framework contained specific patterns • No specific design patterns were chosen • Microsoft framework ensured good balance of the load
Gatekeeper	• No real gatekeepers • Lead designer communicated design decisions to other designers • Lead analyst communicated changed requirements and aggregates information from other analysts • Architect in India needed (linking pin for designers regarding technical documentation)
Reduce ambiguity	• Use cases contained ambiguity • Domain knowledge was sufficient due to glossary • Document all relevant nouns in glossary document
Defining quality requirements	• Non-functionals not specified, assumptions were made • Estimations about performance difficult • Unclear specification of function caused inefficient search function • Non-functionals should be defined using a metric • Clearly specify the functions of the system with enough details
Enough context of the surroundings	• Authorization and interfaces with external systems was not specified • Missing detailed description of the customer process
Verification of requirements	• Due to time constraints, only 80% of the requirements were reflected in design
Required skills and experience in development team	• There is not a lot of experience of past projects in India and to reuse components • Analysts with experience in market segment of the customer is important • Knowledge about how to reuse components not always present in India
Changing requirements	• User Interface requirements regularly changed • Requirements changed due to unclear process modeling • Chat sessions help to communicate new requirements

Availability of team members	• A lot of holidays in India • No big time difference, Indians start and stop 3,5 hours earlier • By telephone and chat sessions requirements were communicated
Team Dynamics	• Team behaves less creative in India due to higher CMMI level, which causes the need to make all requirements and design decision explicit • Indians should stay developing at one specific part of the system where they already gained knowledge • Having the same CMMI-level at both countries is difficult, but would make it easier, as the same level of detail is needed in Artifacts.
Cultural differences	• Mistakes are sometimes not addressed to the right category in order to satisfy CMMI statistics • Indian team does not introduce new ideas • Indian team does not act as dominant member of the group • Getting work done is most important than quality, which indicates short term thinking • Dutch people too direct for Indians • Communication training gives better understanding • Indians cannot dealt with uncertainty and need to have all details during implementation

5.7.3 Results Case Study “Bravo”

Attribute	Findings
Minimum requirements documents needed for design	• Needed artifacts in RUP: Vision, Storyboard, Use Case Model, Use Case Descriptions, Supplementary Specifications, Glossary
Level of design needed in development location	• Artifacts needed of analysis and design: Architectural proof of concept, Data Model • Missing design documents: Analysis model, Design Model, Navigation maps and deployment model • Technical design documents should be specified at detail level to ensure Indian developers know what to build and no misinterpretation occurs • Indian team checks design whether it meets all requirements criteria
Reuse of building blocks	• Microsoft framework was used and there was training in C#, Visual studio and 3rd party tools • Data access layer, user controls and business classes components were reused and proper documentation of these components is needed • One code base showed problems during implementation immediately • Bringing Indian developers to The Netherlands helps to transfer tacit knowledge
Consistent information	• Inconsistent use of words resulted in unclear specifications • Use cases were checked by peer-groups, customers and testers • Glossary in Dutch and English results in better understanding at both sides • Important artifacts to check: Use cases, Storyboard and Design model • Design review in India will detect main consistency flaws
Analysis and design patterns	• No patterns were explicitly chosen and resulted problems during implementation, as there was little structure for the builders of the application • Application data block pattern was used of Microsoft for data access performance

Gatekeeper	• No gatekeeper • One requirement specier available for questions • Information in Clearquest difficult to maintain • To improve knowledge of Indian Team, it is useful to let them come to The Netherlands
Reduce ambiguity	• Use cases, Vision & Scope and Supplemental Specifications were unclear • Translation of use cases done by agency was unsuccessful • Not enough contextual information for Indian developers • Contextual information communicated verbally was misinterpreted • More details about customer should be specified in use cases • Glossary is important in order to deal with ambiguity • Documentation of the customer workflow is important for developers and doing reviews reduces ambiguity of these documents.
Defining quality requirements	• No quality requirements were documented for the first subproject, as the architecture already existed • No metrics were used for the third subproject, which resulted in estimation errors and can ultimately cause performance problems of the system
Enough context of the surroundings	• The problem domain was not clear for developer • Indian developers unfamiliar with customer processes • The configuration of the environment at the customer is important for testers so they can test in the same environment
Verification of requirements	• All requirements specified were reflected in design • Not all requirements were implemented correctly due to misunderstandings • For the third subproject, a requirement of the customer was misunderstood in analysis phase and not reflected in the design
Required skills and experience in development team	• Experience with RUP model and how to use it is important • Decent English speaking and writing skills are needed • Developer should be aware of business processes of customer • Listening to other team members is important • Knowledge about CMMI processes is desired
Changing requirements	• Process requirements changed (new laws changed the process requirements) • Presentation requirements changed most frequently (GUI, Style guide) • Not enough usability experts at LogicaCMG to deal with User Interface issues • A change is implemented after the initial set of requirement is implemented first, which ensures the whole development is not disturbed
Availability of team members	• In October there is low availability due to many holidays • Working during the weekends is no problem in India • Time difference has advantage that Indian are not bothered with the changes in the code at the end of the day, as they start and stop earlier • Communication media was not used enough according to Dutch respondents • Having a good project plan that takes care of the availability of team members ensures that availability is no issue • Indian developer indicated that e-mail, office communicator and other communication means ensures that the team is well connected

Team dynamics	• **No direct communication makes the team less dynamic and causes interpretation problems** • **The forming process of the Team does not take place** • **Most communication is done by means of a chat application** • **Conference calls can be difficult, as it is difficult to understand Indians due to language problems**
Cultural differences	• **Indian developer listen mostly to more powerful members in an organization and need explicit directions to get work done** • **During changes of the system, Indian developers do not ask a lot of questions** • **Indian team members were seen as strangers** • **Indian developers have trouble coping with uncertainty** • **Dutch team member were more open-minded about other opinions**

5.7.4 Results Case Study "Charlie"

Attribute	**Findings**
Minimum requirements artifacts needed for design	• **Needed artifacts in RUP: Vision, Use Case Model, Supplementary Specifications (FURPS, ISO 9126), Prototype of important screens, Glossary**
Level of design needed in development location	• **Artifacts needed of analysis and design: SAD, Analysis model and Interface document** • **Design and development shipped to India after crucial components implemented in The Netherlands** • **Straightforward components that are less crucial for the system were designed and implemented in India**
Reuse of building blocks	• **Login components from bank were used** • **From LogicaCMG, configuration application was reused** • **Spring framework was used, but unknown for Indians, which resulted in understanding problems** • **Indian developers mainly doing plain java work** • **For a bigger project an architect in India would be very useful in order to communicate technical details**
Consistent information	• **Interfaces of the system not consistently specified** • **Behaviour of the system was not clearly specified and did not match (Events and actions)**
Analysis and design patterns	• **Used patterns are: MVC, Façade, Transfer object, Singleton and abstract factory** • **The Spring framework contains patterns like MVC, Proxy, Singleton abstract factory** • **Spring framework made call of interfaces very easy and there was high flexibility of dependencies between objects** • **Use of Party Analysis Pattern to model abstraction**
Gatekeeper	• **No real gatekeeper** • **The lead developer that had to communicate the requirements to Indian developers failed, as there were still misunderstandings**

Reduce ambiguity	• Business logic was missing that describes the services available • Contextual information was difficult to transfer to developer using a chat application • Translation agency caused many bad translations that introduced ambiguity of the Use Cases • A glossary helps to deal with ambiguity
Defining quality requirements	• The supplemental specification contained wrong estimations • Efficiency estimations of the transaction were difficult to make (number of transaction turned out to be twice as high)
Enough context of the surroundings	• List of services available and a clear interface document helps to grasp the environmental context
Verification of requirements	• Almost all requirements were reflected in design • The requirement to make reports of the transactions was dropped due to time constraints
Required skills and experience in development team	• Skills and experience with Spring Framework and data storage possibilities • Résumé Indian developer did not reflect real situation
Changing requirements	• The requirements relating to the interfaces of the system changed • Changes mainly done in The Netherlands • Documents are not always updated after a change
Availability of teammembers	• Not really an issue • No big time difference • Proper project planning tackles this issue
Team dynamics	• Lead Developer from India was in The Netherland during the inception phase to gain contextual information about the customer • There were still misunderstandings in India, as the Lead Developer did not communicate all relevant information to the developers • No reuse of code between Indian developers • The Lead Developer did not intervene to improve communication efforts
Cultural differences	• Making remarks about Indian developers dangerous, as they are easily offended • Some Indian developers do not always listen and can be quite stubborn • If Indian developers do not understand, they do not quickly indicate this

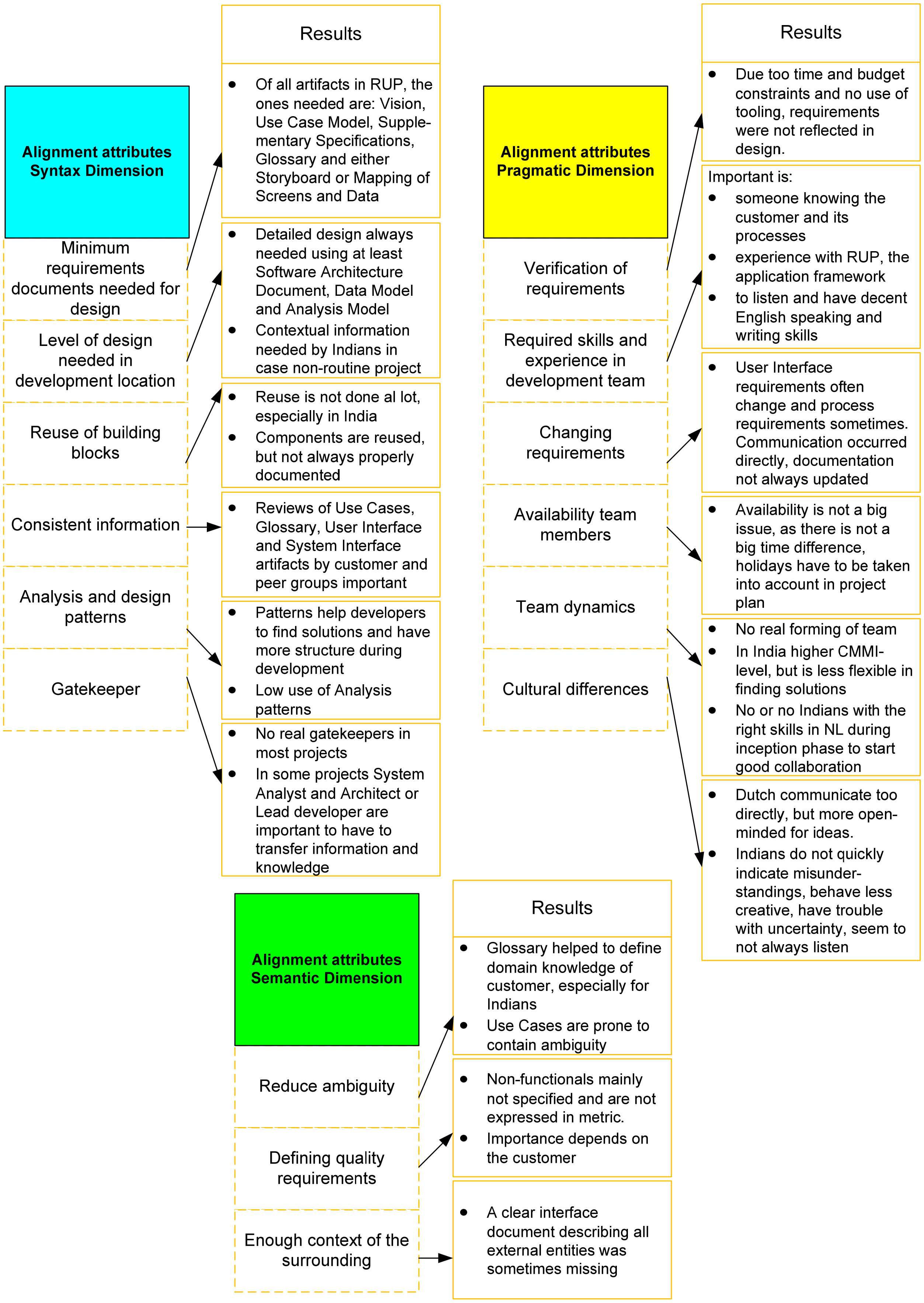
5.7.5 Results Case Study “Delta”

Attribute	Findings
Minimum requirements artifacts needed for design	• Needed artifacts of RUP: Vision, Use Case Model, Use Case Realizations, Supplementary Specifications, Glossary
Level of design needed in development location	• Artifacts needed of analysis and design: SAD and all technical artifacts in RUP • Strategy to do all technical design in The Netherlands to limit risks • Detailed design is possible in India if there is a good understanding of the whole project, the functional design and the abstract technical design • Bringing Indian developers during inception phase to The Netherlands helps to exchange information and knowledge related to design

Reuse of building blocks	• Security component used from external party and explained using direct communication • Dutch development team only used the component • Training sessions help to understand the components
Consistent information	• Most inconsistencies detected during reviews • During design, remaining inconsistencies were noticed and solved immediately
Analysis and design patterns	• MVC pattern
Gatekeeper	• No gatekeeper or single point of contact • There was one Indian Lead Designer that came to The Netherlands to understand the context of the project
Reduce ambiguity	• Lack of information and knowledge of Financial domain • Someone with proper domain-level knowledge helps to reduce ambiguity
Defining quality requirements	• Number of hours for each function point took more time, as Java is being used • Security in Supplementary Specifications was not high enough
Enough context of the surroundings	• External entities were clearly described in Interface document • SAD contained enough context of the environment
Verification of requirements	• No tool used to manage requirements made it difficult to do verification • Use cases can be used for verification of functional requirements, but this was too time-consuming
Required skills and experience in development team	• Customer unfamiliar with RUP • Analyst have to speak the customers' language to properly explain RUP • Domain knowledge of psychiatry important • RUP essentials training important for main contact person at customers' site
Changing requirements	• Preliminary version showed that user interface needed to be changed • The level of security changed
Availability of team members	• Proper planning deals with availability issues • Holidays have to be taken into account in project plan
Team dynamics	• Letting Indian developers come to The Netherland will decrease tension, improve cooperation and communication between team members • The English skills of team members was inadequate and caused misunderstandings
Cultural differences	• Indian team has stronger team spirit, but is less time-efficient due to strong social interaction • Job switching common in India and causes loss of experience

5.8 Overview Results All Case Studies

Figure 22 shows the findings of the case studies at abstract level. Below each color there are quality attributes of the alignment framework and to the right side, the results are shown based on analysis of the case study results.



Alignment attributes Syntax Dimension	Results
Minimum requirements documents needed for design	• Of all artifacts in RUP, the ones needed are: Vision, Use Case Model, Supplementary Specifications, Glossary and either Storyboard or Mapping of Screens and Data
Level of design needed in development location	• Detailed design always needed using at least Software Architecture Document, Data Model and Analysis Model • Contextual information needed by Indians in case non-routine project
Reuse of building blocks	• Reuse is not done al lot, especially in India • Components are reused, but not always properly documented
Consistent information	• Reviews of Use Cases, Glossary, User Interface and System Interface artifacts by customer and peer groups important
Analysis and design patterns	• Patterns help developers to find solutions and have more structure during development • Low use of Analysis patterns
Gatekeeper	• No real gatekeepers in most projects • In some projects System Analyst and Architect or Lead developer are important to have to transfer information and knowledge

Alignment attributes Pragmatic Dimension	Results
Verification of requirements	• Due too time and budget constraints and no use of tooling, requirements were not reflected in design.
Required skills and experience in development team	Important is: • someone knowing the customer and its processes • experience with RUP, the application framework • to listen and have decent English speaking and writing skills
Changing requirements	• User Interface requirements often change and process requirements sometimes. Communication occurred directly, documentation not always updated
Availability team members	• Availability is not a big issue, as there is not a big time difference, holidays have to be taken into account in project plan
Team dynamics	• No real forming of team • In India higher CMMI-level, but is less flexible in finding solutions • No or no Indians with the right skills in NL during inception phase to start good collaboration
Cultural differences	• Dutch communicate too directly, but more open-minded for ideas. • Indians do not quickly indicate misunderstandings, behave less creative, have trouble with uncertainty, seem to not always listen

Alignment attributes Semantic Dimension	Results
Reduce ambiguity	• Glossary helped to define domain knowledge of customer, especially for Indians • Use Cases are prone to contain ambiguity
Defining quality requirements	• Non-functionals mainly not specified and are not expressed in metric. • Importance depends on the customer
Enough context of the surrounding	• A clear interface document describing all external entities was sometimes missing

Figure 22 - Abstract overview of case study results

5.9 Importance of Quality Attributes

The importance of the quality attributes for alignment was asked to several respondents that were available. The results are shown in table 21, where the most important quality attributes are listed first. For prioritizing the importance of the attributes we have only looked at the arithmetic mean.

According to the respondents the six most important quality attributes are *"Consistent information"*, *"Reduce Ambiguity"*, *"Required skills and experience in development team"*, *"Verification of requirements"*, *"Minimum requirements documents needed for design"* and "*Gatekeeper*". The attributes less important are "*Availability team members*" and *"Analysis and design patterns"*.

Table 21 - Importance of quality attributes

Alignment Attributes	Mean	St. Dev.	Median	Range	Mode
Consistent information	3,40	0,84	4	2	4
Reduce Ambiguity	3,40	0,52	3	1	3
Required skills and experience in development team	3,30	0,82	3,5	2	4
Verification of requirements	3,30	0,48	3	1	3
Minimum requirements documents needed for design	3,30	0,82	3,5	2	4
Gatekeeper	3,10	0,74	3	2	3
Changing requirements	3,10	0,99	3	3	4
Level of design needed in development location	3,00	0,82	3	2	3
Cultural differences	2,90	1,10	3	3	3
Defining quality requirements	2,90	0,74	3	2	3
Reuse of building blocks	2,80	0,63	3	2	3
Enough context of the surroundings	2,60	0,84	3	3	3
Team dynamics	2,60	0,70	3	2	3
Analysis and design patterns	2,10	0,74	2	2	2
Availability team members	2,10	0,74	2	2	2

Importance (Scale: 1 - 4)

⬜	1
🟧	2
🟧	3
🟥	4

Appendix G shows the results from the respondents. The mean, standard deviation, median, range and mode can be seen in table 21. The arithmetic average of a set of values is often simply called the "mean" ($\bar{x}$). The mean is calculated with the following formula:

$$\bar{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

The letter N is the number of samples taken, which is in this case ten. The standard deviation (σ) is a measure of how widely values are dispersed from the average value (the mean $\bar{x}$). The standard deviation can be calculated with the following formula:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2}.$$

The median of a finite list of numbers can be found by arranging all the observations from lowest value to highest value and picking the middle one. When there are an even number of observations, the median is not unique, so one often takes the mean of the two middle values. The range is the length of the smallest interval which contains all the data and is calculated by subtracting the smallest observations from the greatest and is an indication of the statistical dispersion. The mode is the most frequent value that is given in a particular sampling.

5.10 OVERVIEW OF THEORY AND PRACTICE

Several quality attributes for alignment are recognized in theory, as well as in practice. Table 22 shows which attribute can be found in theory and practice. The attributes that were found in practice, but not in theory are highlighted green. The attributes found in theory, but not in practice are highlighted in red.

Table 22 - Attributes in theory and practice

Found in / Attribute	Theory		Practice
Consistent information	✓	(IEEE-Std.830 1998), (Lauesen, S. 2002),	✓
Reduce Ambiguity	✓	(IEEE-Std.830 1998), (Lauesen, S. 2002), (Kosman, R.J. 1997)	✓
Required skills and experience in development team	×		✓
Verification of requirements	✓	(IEEE-Std.830 1998), (Lauesen, S. 2002)	✓
Minimum requirements documents needed for design	✓	(IEEE-Std.830 1998), (The Standish Group, International 2001), (Lauesen, S. and Vinter, O. 2001)	✓
Gatekeeper	×		✓
Changing Requirements	✓	(IEEE-Std.830 1998), (Hoorn, J.F. et al. 2007), (Alves, C. and Finkelstein, A. 2002), (Herbsleb, D. and Mockus, A. 2003), (Nuseibeh, B. and Easterbrook, S. 2000).	✓
Level of design needed in development location	×		✓
Cultural differences	✓	(Damian, D. and Zowghi, D. 2003), (Hofstede, G. 1994)	✓
Defining quality Requirements	✓	(Guha, R. et al. 2004), (The Standish Group, International 2001), (Gilb, T. 1997), (Lauesen, S. 2002), (Kosman, R.J. 1997)	✓
Reuse of building blocks	×		✓
Enough context of the surroundings	✓	(Lauesen, S. 2002)	✓
Team dynamics	×		✓
Analysis and design patterns	✓	(Gross, D. and Yu, E. 2001), (Fowler, M. 1997), (Gamma, E. et al. 1995), (Buschman, F. et al. 2007), (Lasater, C.G. 2006)	×
Availability team members	✓	(Damian, D. and Zowghi, D. 2003)	×

To the best of our knowledge there are no papers found that are important for alignment of requirements and design for the attributes *required skills and experience in development team, gatekeeper, level of design needed in development location, reuse of building blocks* and *team dynamics.*

Alignment attributes like *consistent information, reducing ambiguity, verification of requirements, minimum requirement documents needed,* and *changing requirements* have been recognized in theory and practice and have even been embedded standards like the IEEE standard for specifications (IEEE-Std.830 1998). These attributes are imperative in order to properly communicate the requirements to the designers of the system, avoid misunderstandings and make changing requirements easier to maintain.

Required skills and experience, *gatekeeper* and *level of design needed in development location* were not found important in theory. However, in practice these attributes were recognized as very important. The people in a development team are the key to success and if you have the right skills and experience and the right level of design, this makes it possible to properly design the system under development. In addition, gatekeepers are important to have in your development team to ensure someone takes the responsibility to communicate functional and technical information. Furthermore, the interview results show that the attributes *reuse of building blocks* and *team dynamics* should be considered important.

In practice, *analysis and design patterns* and *availability team members* were not found very important for alignment of requirements and design and this differs from the found theory in this research. Availability of team members was not found important, as the small time difference in India and The Netherlands did not cause any real problems. Analysis and design pattern were not explicitly used in most projects, although a lot of design pattern are imbedded in the frameworks that are being used at LogicaCMG. Only one case study showed the appliance of an analysis pattern to transfer knowledge about the customers' processes.

5.11 SUMMARY

By means of several case studies, information has been gathered to see how alignment of requirements and design is done at LogicaCMG. The alignment framework of chapter four has been applied to each project and by means of interviews and documentation information has been gathered about each of the quality attributes for alignment. The framework in chapter four has been refined by the findings of the four case studies. The quality alignment attributes have been validated by doing the several case studies and looking whether the attributes were important during the project and the reasoning behind this. Figure 22 shows an overview of the case studies results. At the end of this chapter the quality attributes were prioritized by different respondents. The top six attributes are *"Consistent information", "Reduce Ambiguity", "Required skills and experience in development team", "Verification of requirements", "Minimum requirements documents needed for design"* and *"Gatekeeper"*.

6 TOWARDS IMPROVED ALIGNMENT OF REQUIREMENTS AND DESIGN

6.1 INTRODUCTION

The previous chapter shows the appliance of the assessment framework. The results of the interviews have been presented for each case study and showed whether the quality attributes are important in practice.

This chapter defines the guidelines for each quality attribute that can be used to improve the alignment of requirements and design in the blended delivery model. These guidelines came up during the interviews performed for the case studies. Furthermore, experts at LogicaCMG gave me advice regarding the different quality attributes and the related guidelines. This chapter presents the guidelines that can easily be used as a kind of checklist during the inception and elaboration phase of RUP. The guidelines are categorized into general and the two disciplines requirements and analysis and design, as these are the two disciplines in the RUP model where this research applies to. The structure of the guidelines is depicted in figure 23. To the left, indicated by the light blue color, the goal of the quality attributed is stated. To the right, the guidelines can be found, which is indicated by a white box. The arrows correspond to the different guidelines for each quality attribute of the alignment framework.

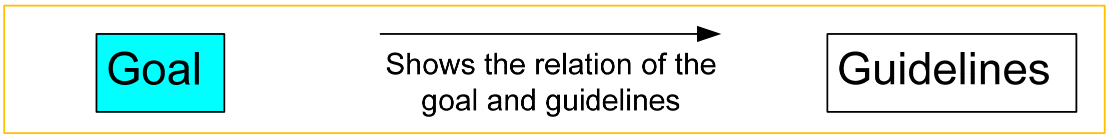


Figure 23 - Structure of the guidelines

6.2 GENERAL GUIDELINES

In this paragraph general guidelines are listed that can be applied during all the phases of a RUP project. The guidelines are derived from the case studies described in the previous chapter. These guidelines help to achieve the goal of the quality attributes of the alignment framework.

6.2.1 Gatekeeper

To improve information and knowledge exchange, two persons are needed that communicate either, the functional or technical design to the right communication channels. A lead developer or architect in India needs to be assigned to communicate the technical design. It is important that this gatekeeper is accepted by the other Indian developers in order be sure all important messages are received properly and not lost due to psychological interference, as explained in paragraph 3.3.1. To improve the understanding of the Indian Lead Developer, it is good to bring this person to The Netherlands to transfer tacit knowledge between team members and be more acquainted with the customer. Furthermore a Lead Analyst is needed, who is mainly responsible for communication of the functional design to the designers.

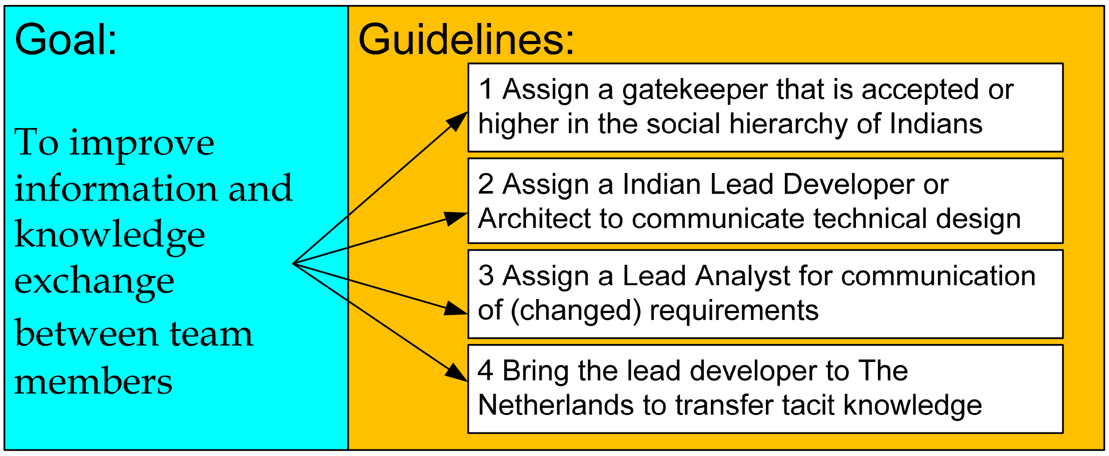

Goal:
To improve information and knowledge exchange between team members
Guidelines:
1 Assign a gatekeeper that is accepted or higher in the social hierarchy of Indians
2 Assign a Indian Lead Developer or Architect to communicate technical design
3 Assign a Lead Analyst for communication of (changed) requirements
4 Bring the lead developer to The Netherlands to transfer tacit knowledge

Figure 24 - Guidelines Gatekeeper

Recommendation 1: Assign two gatekeepers responsible for communication of either the functional or technical design and bring the Lead Developer to The Netherlands to transfer tacit knowledge.

6.2.2 Required skills and experience in development team

Having the right skills and experience in your development team is important during development. An analyst in your team that knows the customers' processes helps to make a functional design that really fits the needs of the customer. In addition, proper training of Indian developers related to regularly used components is needed to enhance the skills of your development team. Proper training of the RUP development model and English writing skills should be given to analysts, developers and the main point of contact from the customer when these skills are inadequate. Figure 25 shows the guideline for this quality attribute.

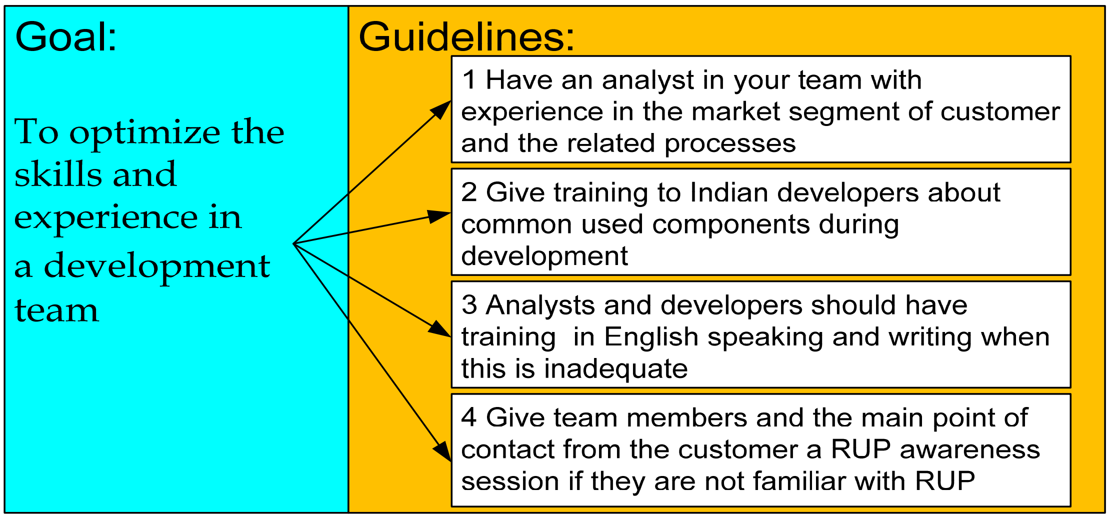

Goal:
To optimize the skills and experience in a development team
Guidelines:
1 Have an analyst in your team with experience in the market segment of customer and the related processes
2 Give training to Indian developers about common used components during development
3 Analysts and developers should have training in English speaking and writing when this is inadequate
4 Give team members and the main point of contact from the customer a RUP awareness session if they are not familiar with RUP

Figure 25 - Guidelines Required skills and experience in development team

Recommendation 2: Give training in English, RUP and commonly used components to the development team.

6.2.3 Availability team members

Availability is not really an issue if the project plan takes into account the different holidays in India. Communication tools like chat, telephone and videoconferencing can be regularly to communicate with each other and enhance the availability of team members. Figure 26 shows the guidelines to prevent availability problems of Indian team members.

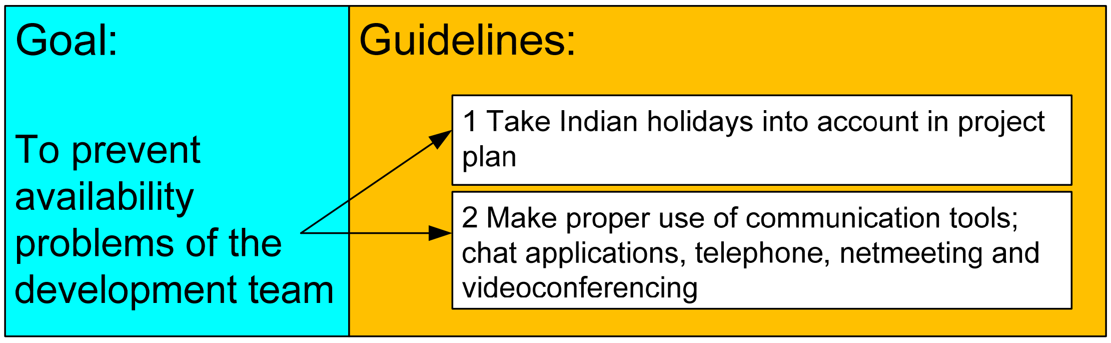

Goal:
To prevent availability problems of the development team
Guidelines:
1 Take Indian holidays into account in project plan
2 Make proper use of communication tools; chat applications, telephone, netmeeting and videoconferencing

Figure 26 - Guidelines Availability team members

Recommendation 3: Take Indian holidays into account in project plan and make proper use of available communication tools.

6.2.4 Team dynamics

In order to have interactive collaboration, it is important that the Indian developers know their colleagues in The Netherlands. For this reason it is important to bring at least the Lead Developer to The Netherland to break the barrier between the two distributed groups. A kick-off meeting with the whole team is important to start the fundament for collaboration between team members (Tuckman, B. 1965). During the kick-off meeting different questions can be answered about the project and team member can get more acquainted with each other. The communication tools available should be regularly used to share ideas and exchange information with each other. No new recommendation is needed, as recommendation three covers the proper use of available communication tools and recommendation one already talks about bringing the Indian Lead Developer to The Netherlands. Figure 27 shows the guidelines for group dynamics.

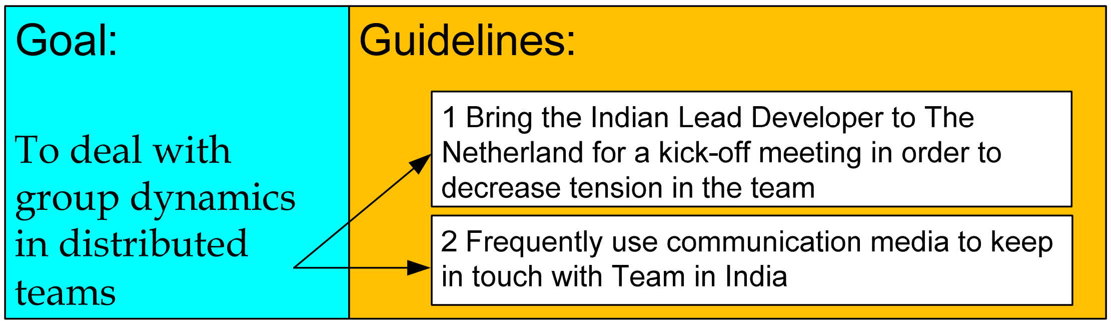

Goal:
To deal with group dynamics in distributed teams
Guidelines:
1 Bring the Indian Lead Developer to The Netherland for a kick-off meeting in order to decrease tension in the team
2 Frequently use communication media to keep in touch with Team in India

Figure 27 - Guidelines Team dynamics

6.2.5 Cultural differences

There are many cultural differences between the Indian and Dutch team members. In order to deal with the differences, it is important that Dutch team members are less direct during communication to Indian developers and try to formulate remarks in a subtle way.

Indian developers might be offended if they get in their opinion direct (blunt) remarks and ultimately might refuse to work. Communication training helps to cope with these different ways to communicate and make the difference as small as possible. As a final point, Indian developers should be given detailed instructions about the work that has to be done. Feedback of Indian developers should always be asked, as it is not always clear whether they understand the functional and technical design documents. Figure 28 shows the guidelines for cultural differences.

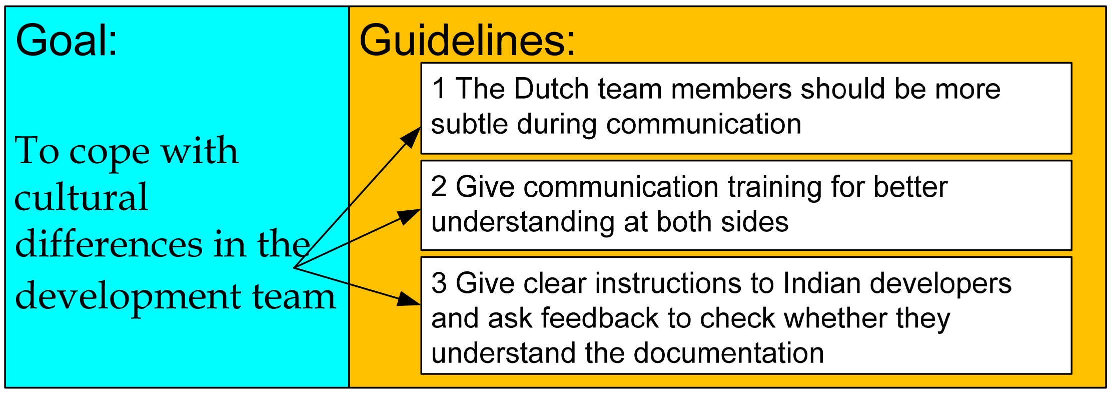

Goal:
To cope with cultural differences in the development team
Guidelines:
1 The Dutch team members should be more subtle during communication
2 Give communication training for better understanding at both sides
3 Give clear instructions to Indian developers and ask feedback to check whether they understand the documentation

Figure 28 - Guidelines Cultural differences

Recommendation 4: Team members should have a cultural training about how to communicate with each other and Indian developers should be given clear instructions regarding their work.

6.3 REQUIREMENTS GUIDELINES

In this paragraph several guidelines are described that can used for the requirement discipline of a RUP project. The guidelines are derived from the case studies described in the previous chapter.

6.3.1 Minimum requirements documents needed for design

There are specific requirements artifacts in RUP that have to be defined in any project. The requirements artifact needed for design are Vision, Use Case Model, Glossary and Supplementary Specifications that clearly describes architectural requirements of the system by using FURBS, as described in paragraph 3.6.3. As RUP is being used, Use Cases are automatically chosen to serve as functional requirement specification technique. The whole RUP process uses this requirement technique and for this reason, it is better to keep using this technique. Furthermore, the tooling that can be used like RequisitePro for requirement management is also adapted only for this technique. For defining non-functional requirements, Planguage (Gilb, T. 2005) can be used to clearly makes use of metrics.

In addition, either Mapping of Screens and Data or Storyboard is important to describe the user interface details. User interface issues should not be described in use cases to keep them clear and easy readable. Moreover, to make the life of designers easier these two artifacts should be put in one document. This way, the designer can immediately see the functional requirements in the use cases and the description of the user interface in one document. Figure 29 shows the guidelines for this quality attribute of the alignment framework.

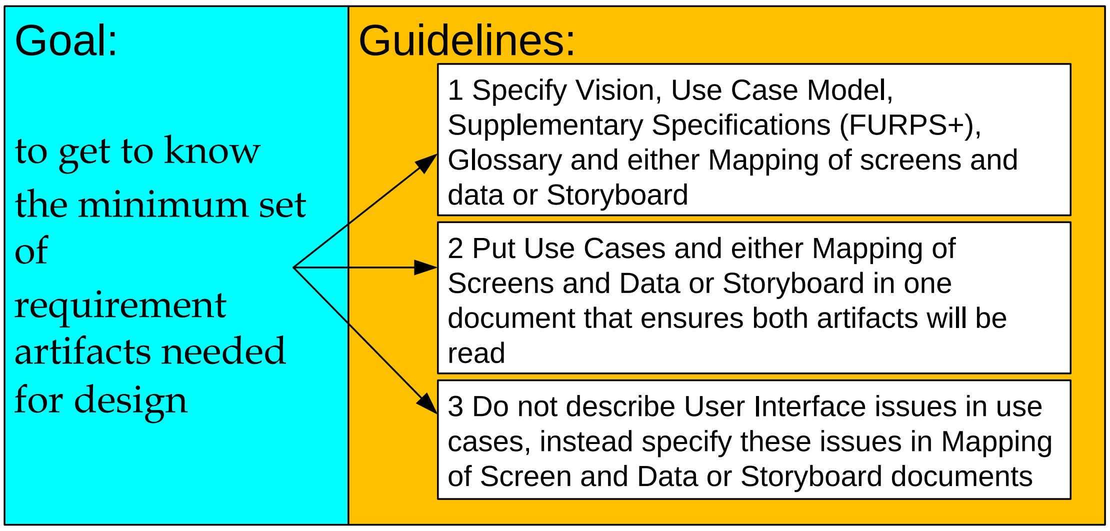

Goal:
to get to know the minimum set of requirement artifacts needed for design
Guidelines:
1 Specify Vision, Use Case Model, Supplementary Specifications (FURPS+), Glossary and either Mapping of screens and data or Storyboard
2 Put Use Cases and either Mapping of Screens and Data or Storyboard in one document that ensures both artifacts will be read
3 Do not describe User Interface issues in use cases, instead specify these issues in Mapping of Screen and Data or Storyboard documents

Figure 29 - Guidelines Minimum requirements documents needed for design

Recommendation 5: For each project specify at least the following requirement artifacts: Vision, Use Case Model without describing User Interface issues, Supplementary Specifications, Glossary and either Mapping of Screens and Data or Storyboard.

6.3.2 *Consistent information & Reduce ambiguity*

The quality of artifacts can be improved by proper reviewing of the customer, a tester and builder to make sure there are no inconsistencies in the artifacts from technical and functional point of view. Important artifacts to check are: Use Cases, Glossary, User Interface and System Interface artifacts, as these artifacts are prone to contain inconsistencies. Making a Glossary that describes all relevant nouns helps to use only one kind of word for the same thing and that these words are used consistently in all artifacts.

Ambiguity is one of the reasons why requirements are not reflected in design. The artifacts that are prone to contain ambiguity are: Use Cases, Vision and Supplementary Specifications. In order to prevent different interpretations, it is important to specify all relevant nouns in the Glossary artifact. Furthermore, translation of artifacts should be done by the person who is responsible for it. For Use Cases these persons are mainly the requirement specifiers. This prevents that IT terminology is wrongly used and that no important requirements are lost in translation. The guidelines for this attribute are the same as the *"Consistent information"* attribute. Figure 30, shows the guidelines for these alignment attributes.

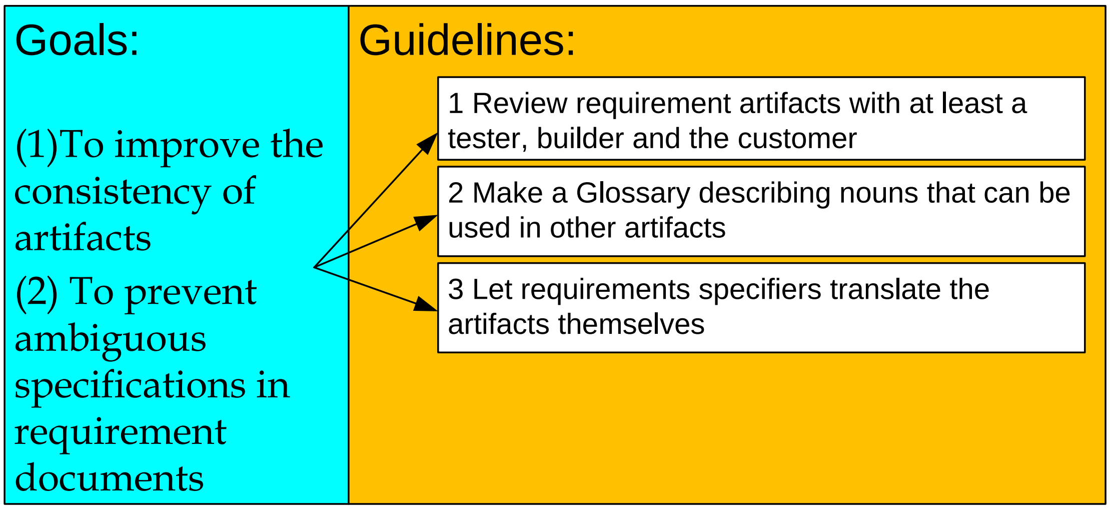

Goals:
(1)To improve the consistency of artifacts
(2) To prevent ambiguous specifications in requirement documents
Guidelines:
1 Review requirement artifacts with at least a tester, builder and the customer
2 Make a Glossary describing nouns that can be used in other artifacts
3 Let requirements specifiers translate the artifacts themselves

Figure 30 - Guidelines Consistent information and Reduce ambiguity

Recommendation 6: To improve consistency and reduce ambiguity, review artifacts with at least a tester, a builder and the customer, define all relevant nouns in the Glossary artifact and let requirement specifiers translate the artifacts into English.

6.3.3 Verification of requirements

In order to be able to verify whether the product components meet the specified requirements, it is important to look at two main guidelines. First of all, unrealistic requirements should be dropped immediately, which results in a requirement set that is easier to maintain. To achieve this, the documents need to be properly reviewed during an intake by the designers. This guideline is already stated previously and will not be repeated in figure 31. Furthermore, requirement management tooling is needed to maintain the requirements of a system. Properly using tools like Optimal Trace or RequisitePro makes verification easier and ensures that no important requirements are forgotten.

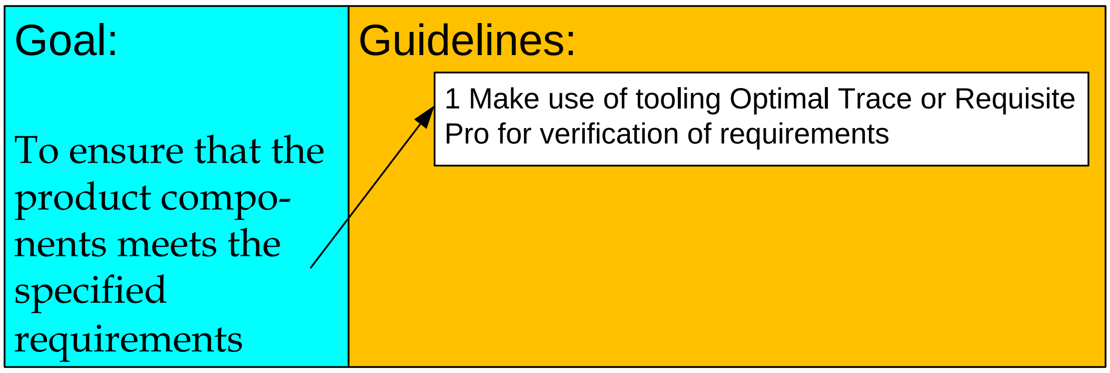

Goal:
To ensure that the product compo-nents meets the specified requirements
Guidelines:
1 Make use of tooling Optimal Trace or Requisite Pro for verification of requirements

Figure 31 - Guidelines Verification of requirements

Recommendation 7: Make use of proper requirements management tooling, like Optimal Trace or RequisitePro to ensure the product meets the specified requirements.

6.3.4 Changing requirements

During the development requirements related to the User Interface and customer process regularly change. Making a prototype or showing a preliminary version of the product is useful to see how the application should look like and identify wrong User Interface requirements. Artifacts related to the changed requirements should be updated immediately, as this is frequently forgotten. As a final point, proper tooling like RequisitePro makes it easier to cope with changed requirements. This way, dependencies are clear and artifacts are always updated after a requirements change. Figure 32 shows the guidelines to cope with changing requirements in artifacts.

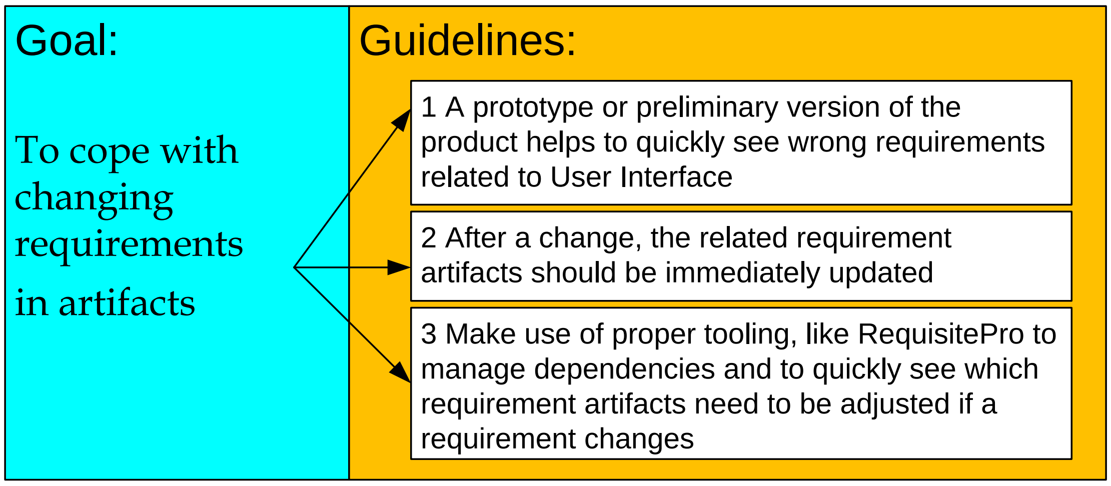

Goal:
To cope with changing requirements in artifacts
Guidelines:
1 A prototype or preliminary version of the product helps to quickly see wrong requirements related to User Interface
2 After a change, the related requirement artifacts should be immediately updated
3 Make use of proper tooling, like RequisitePro to manage dependencies and to quickly see which requirement artifacts need to be adjusted if a requirement changes

Figure 32 - Guidelines Changing requirements

Recommendation 8: Proper tooling, like RequisitePro needs to be used to manage requirements artifacts and a prototype or preliminary version of the product needs to be used to identify wrong User Interface requirements.

6.4 ANALYSIS AND DESIGN GUIDELINES

In this paragraph guidelines are defined that can used for the analysis and design discipline of a RUP project. The guidelines are derived from the case studies described in the previous chapter.

6.4.1 Level of design needed in development location

The level of design needed in development locations depends on the kind of tasks that are performed. More contextual information is needed when groups are facing non-routine tasks. Individuals performing routine tasks already know the problem, causal linkages and the decisions that need to be made. In this case, disagreements over preferences are less prominent (Weick, K.E. and Meander, D.K. 1993). Tasks of team members can be described as non-routine for several reasons. The individual may have a new role to perform, is new to working in distributed teams or the task that has to be done is entirely new. So if the task is non-routine, the communication strategy should be to not only communicate the content, but also the context (Majchrzak, A. et al. 2005). Furthermore, a more detailed design ensures Indian developers exactly know what to do and prevents misunderstanding about specific solutions. Figure 33 shows the guidelines related to the level of design that is needed for offshoring.

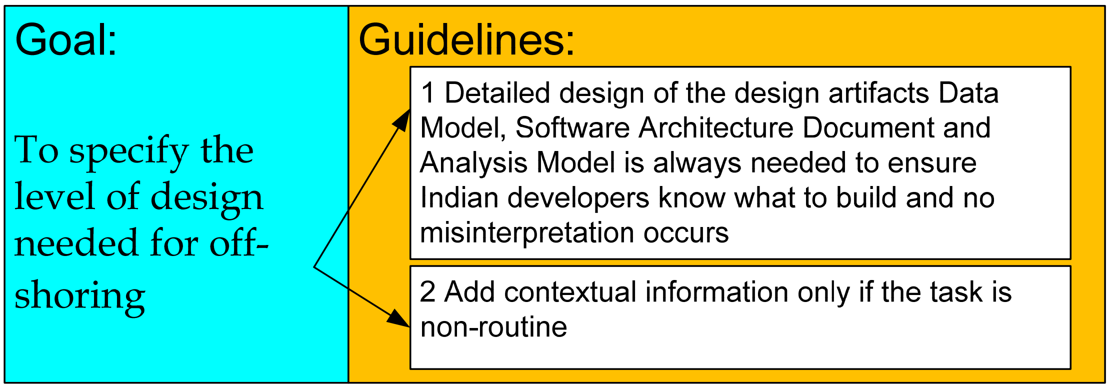

Goal:
To specify the level of design needed for off-shoring
Guidelines:
1 Detailed design of the design artifacts Data Model, Software Architecture Document and Analysis Model is always needed to ensure Indian developers know what to build and no misinterpretation occurs
2 Add contextual information only if the task is non-routine

Figure 33 - Guidelines Level of design needed in development location

Recommendation 9: Contextual information should only be added if the tasks are non-routine and a very detailed design of the Data Model, Software Architecture Document and Analysis Model is needed in the development location to ensure developers know what to do and prevent misinterpretations.

6.4.2 Reuse of building blocks

The first thing important is to try to use building blocks that developers are already familiar with. This prevents startup problems when using the building blocks, as the developers understand how the building block works and do not have to figure out how to apply it. In addition, bringing the Indian Lead Developer to The Netherlands will make it possible that tacit knowledge of the building blocks is transferred between team members. An architect in India is useful for big projects, as he can communicate technical details of the building blocks used and support the developers during the design and implementation of the system. As a final point, the building blocks should be documented properly in order to make reuse straightforward for the developers in India. Bringing the Indian Lead Developer to The Netherlands is already suggested in recommendation one and will not be treated in this recommendation. Figure 34 shows the guidelines to encourage reuse of building blocks.

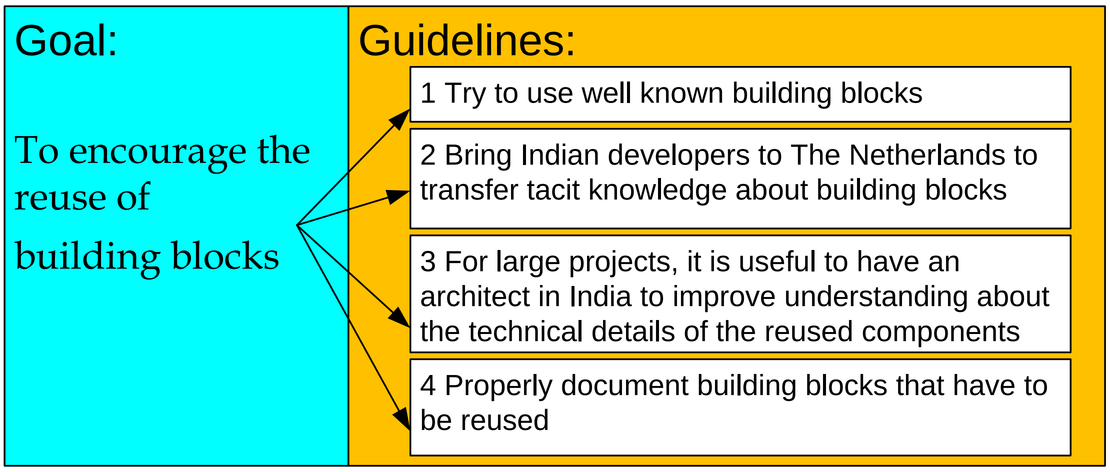

Goal:
To encourage the reuse of building blocks
Guidelines:
1 Try to use well known building blocks
2 Bring Indian developers to The Netherlands to transfer tacit knowledge about building blocks
3 For large projects, it is useful to have an architect in India to improve understanding about the technical details of the reused components
4 Properly document building blocks that have to be reused

Figure 34 - Guidelines Reuse of building blocks

Recommendation 10: To encourage reuse, well known building blocks should be used, building blocks should be properly documented and in case of big projects an architect in India is needed to explain the technical details of the building blocks.

6.4.3 Analysis and design patterns

Knowledge related to a particular solution can be defined in a design pattern. As already said in chapter three, design patterns helps to document and communicate the proven design solution to recurring problems. To efficiently transfer knowledge, it is important that there is common implementation model that all developers in The Netherland as well as in India understand (Lasater, C.G. 2006). Using a design pattern makes all of this possible. Besides design patterns, analysis pattern should be used to define knowledge related to business processes. Figure 35 shows the guidelines to efficiently transfer knowledge using patterns.

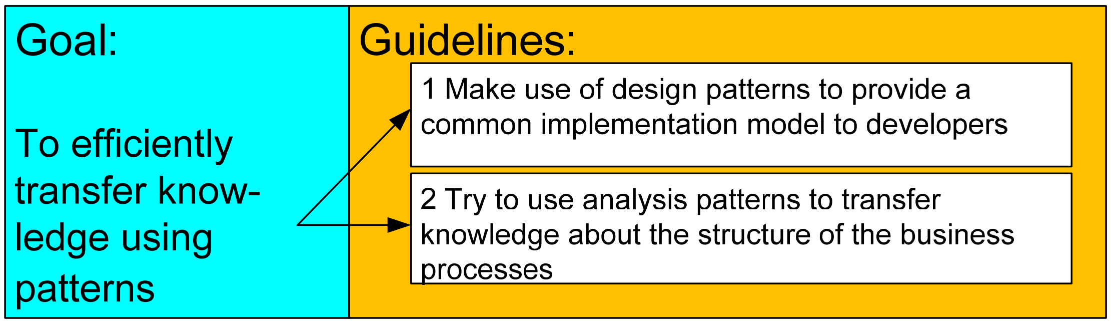

Goal:
To efficiently transfer know-ledge using patterns
Guidelines:
1 Make use of design patterns to provide a common implementation model to developers
2 Try to use analysis patterns to transfer knowledge about the structure of the business processes

Figure 35 - Guidelines Analysis and design patterns

Recommendation 11: Transfer knowledge between team members by using design patterns to provide a common implementation model and analysis patterns to have a better understanding about the structure of the business processes.

6.4.4 Defining quality requirements

Quality requirements are not specified good enough, because no proper tools are used to express them (Simmons, E. 2001) and this can also be seen at LogicaCMG. A realistic estimate of a quality requirement is important for an architect and influences his decision for a specific design decision. It is important that functions are described thoroughly with enough information to properly estimate the quality requirement. To optimize a search function for instance requires that enough information is provided about what to search for. The customer should specify exactly what they want and try to specify a realistic estimate, with help of the system analyst of course. In addition, this can be achieved by using a metric to quantify the non-functional requirements. Planguage is a way to specify quality using keywords and should be used to define important non-functional requirements during a project and to ensure a proper design can be made (Gilb, T. 2005). Figure 36 shows the guidelines when defining quality requirements.

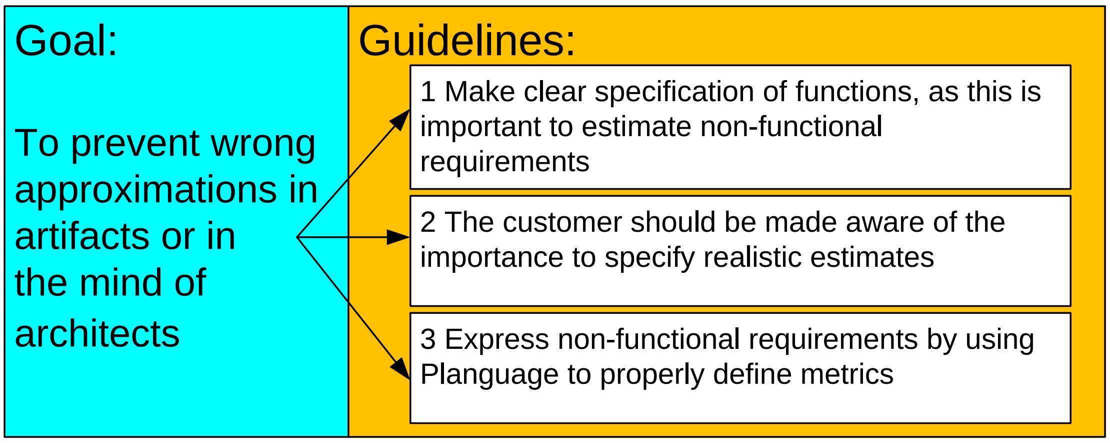

Goal:
To prevent wrong approximations in artifacts or in the mind of architects
Guidelines:
1 Make clear specification of functions, as this is important to estimate non-functional requirements
2 The customer should be made aware of the importance to specify realistic estimates
3 Express non-functional requirements by using Planguage to properly define metrics

Figure 36 - Guidelines Defining quality requirements

Recommendation 12: To prevent wrong estimates, thoroughly specify functions, specify realistic estimates of the system in close cooperation with customer and use Planguage to define metrics of the quality requirements.

6.4.5 *Enough context of the surrounding*

As already explained in paragraph 4.5.3, the surrounding consists of user groups and external systems. Connecting a new system to another system can be very difficult if there is no clear document describing the interfaces of the system with the surrounding (Lauesen, S. 2002). This is also the case at LogicaCMG, as sometimes a clear document describing the interfaces of the system was missing. Figure 37 shows the guidelines to enhance the context of the surrounding in requirements artifacts.

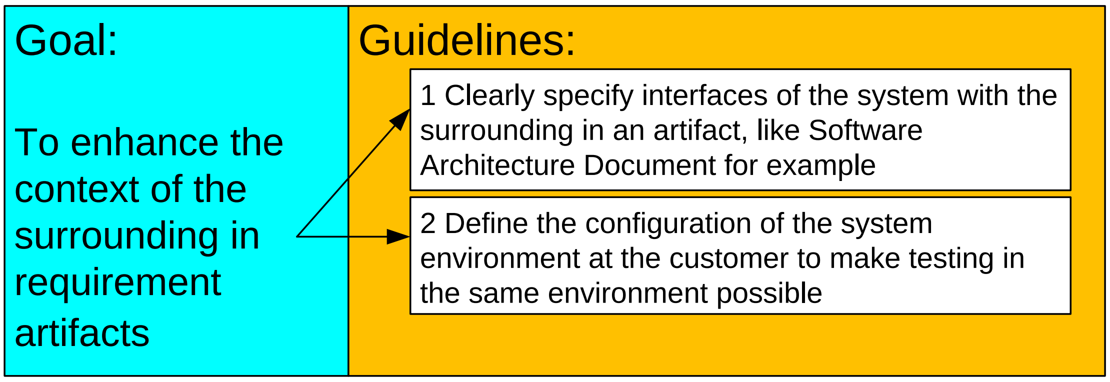

Goal:
To enhance the context of the surrounding in requirement artifacts
Guidelines:
1 Clearly specify interfaces of the system with the surrounding in an artifact, like Software Architecture Document for example
2 Define the configuration of the system environment at the customer to make testing in the same environment possible

Figure 37 - Guidelines Enough context of the surrounding

Recommendation 13: Clearly specify interfaces of the system in an artifact, like Software Architecture Document for example and define the configuration of the system environment in an artifact to make testing in the same environment possible.

7 CONCLUSIONS

In this thesis we tried to discover different concepts that are important to improve the alignment requirements and design during global software development. This chapter starts with explaining what research actions have been taken and showing the results of the research by answering the research questions. Furthermore, an explanation is given about what research results mean. We finish with a reflection about the way the research was performed and discussing interesting topics that can be explored in future research.

7.1 CARRIED OUT ACTIONS

In chapter one, we defined the main problem as *"The lack of alignment between the customer's and developer's site in a multi-department software factory with global software development activities"*. This means that the part where the analyst specifies the requirements and the part where the designer makes the design decisions do not fit with each other. For this reason the following research goal was defined:

To define the quality attributes of the requirements products in a consistent way and to reduce the design problems due to a blended delivery model

In context of this research, quality attributes can improve or worsen the alignment of requirement and design. The constructed framework in this thesis is based on these attributes. The following actions were carried out during this research:

1. Identify the important theoretical alignment concepts; this is needed to learn what we are talking about in the rest of the thesis.
2. Make a framework to assess the alignment during a project; this framework defines the goal of each alignment attribute to improve the alignment in practice and provides key questions that can be asked to the people interviewed for the case studies.
3. Validate the framework to recognize the importance of the quality attributes; several interviews for in depth case studies were conducted to acknowledge the importance of the found attributes.
4. Analyze the alignment concepts in practice by looking at the results of the case studies; this gives us insights about what problems really happen during a project and provides some solutions to cope with the these problems.
5. Provide guidelines to improve the alignment and complement the framework; by looking at the goals of each attribute, several guidelines were formulated that can be embedded in the RUP development method that is used at LogicaCMG.
6. Prioritize the quality attributes of alignment framework; several people filled in the importance of each quality attribute, which ultimately indicates the most important attributes. This was a nice emerging side effect of this research, but was not a goal in the beginning of the research.

7.2 MAIN CONCLUSION

Three research questions were defined to achieve the research goal. For each research question the key answers will be given below.

What are the pitfalls in the alignment of (changing) requirements and design?

The main danger found during alignment is that we are working with people that all have their own preferences of communication during software development. The followed RUP development method has lack of flexibility, as use cases always have to be used as functional requirement style. The way to communicate requirements is often quite different between team members, and depends on culture, kind of documents used, location of the team, the level of design, the experience and skills of team members and so on. Therefore miscommunication is the main danger during alignment of requirements and design. Put it in another way, proper communication is the linking pin between requirements and design.

In a blended delivery environment, what are the quality attributes of the requirement products needed by the designers?

As miscommunication is the main pitfall during alignment, several quality attributes have been categorized into the syntax, semantic and pragmatic dimensions of communication. The importance of the quality attributes was quantified by different respondents, which made it possible to prioritize the attributes. The following quality attributes have been recognized in theory and practice, where the most important attributes are listed first:

1. *Consistent information*
2. *Reduce Ambiguity*
3. *Required skills and experience in development team*
4. *Verification of requirements*
5. *Minimum requirements documents needed for design*
6. *Gatekeeper*
7. *Changing requirements*
8. *Level of design needed in development location*
9. *Cultural differences*
10. *Defining quality requirements*
11. *Reuse of building blocks*
12. *Enough context of the surroundings*
13. *Team dynamics*
14. *Analysis and design patterns*
15. *Availability team members*

How can we define quality attributes for requirements products in order to achieve better alignment of requirement and design in a blended delivery model?

After getting enough information from the respondents of the four case studies, several guidelines were discovered to improve the alignment of requirements and design. Furthermore, experts at LogicaCMG gave me advice about how to improve the alignment during global software development. The guidelines are structured into "*General*" guidelines and the two disciplines *"Requirement"* and *"Analysis & Design"* in RUP. These guidelines can be used in the RUP development process framework of the multi-department software factory at LogicaCMG. The alignment framework has been validated by the respondents of the case studies and experts working at the Result Centre of LogicaCMG and all of them recognized the importance of almost all quality attributes embedded in the framework.

7.3 Implications

The problems during global software development that were identified in previous research are communication, cultural differences, time differences and knowledge exchange. All these problems, except time differences were also acknowledged in this research. An explanation for this is that there is not a real big time difference between The Netherland and India.

The several case study interviews showed very interesting findings. Our results suggest to invest in solving the communication problem by facilitating:

- communication training of different cultures,
- travelling of key communicators,
- training in training in English, RUP and commonly used components,
- forming process of the development team,
- more use of patterns,
- frequent use of communication tools,
- use of proper requirements management tooling .

Communication training of different cultures is needed in order to know how to communicate with each other. Indian developers should be given clear instructions regarding their work and this should be communicated in a subtle way. To communicate functional and technical design, the key communicators like the Indian Lead Developer should travel to The Netherlands for knowledge exchange. Proper training in English, RUP and commonly used components is worth it, as this increases the skills of the development team and prevents misunderstandings. During all the projects of the case studies, Indian and Dutch team members did not collaborate effectively. There was no good interaction of knowledge and mutual learning between team members of these two groups. Both groups are pretending to get on or along with each other and do not let down a politeness barrier. This prevents Dutch and Indian team members to properly exchange knowledge and start sharing ideas about important solutions and how to improve development. The distrust should be taken away by organizing a kick-off meeting where different questions related to the project can be answered and team members can get familiar with each other. To exchange knowledge it is necessary to make use of design and analysis patterns. Design patterns help to communicate technical details, whereas analysis patterns help to communicate the business processes of the customer. To keep all team members in the loop, all communication tools that are available like video conferencing, office communicator and telephone should be used. As a final point, it is imperative to make use of proper requirements management tooling in order to make verification of requirements easier and less time-consuming. This way, no important requirements are forgotten. Another advantage of this tooling is that all documents are updated quite easily if a requirement changes, which basically means that dependencies of requirements are managed in a straightforward way.

7.4 Reflection about the way the research was performed

To improve the alignment of requirements and design, an alignment framework has been developed from theory, experts and my supervisors by (i) looking at different dimensions of communication and finding the related attributes that can improve or worsen the alignment, (ii) validating these attributes by doing several case studies and (iii) providing guidelines to achieve the goal of each alignment attribute that is described in the framework. The performed research presented in this thesis can be described as an explorative research, as there was not a list or complete theory that could be used to test or apply an existing framework. Therefore, this research is somewhat limited, as it is not sure if it really covers all alignment concept related to the research problem identified. The derived framework should be looked into by testing it and refining it iteratively. More subjects for future research are discussed in paragraph 7.5.

7.4.1 Validation and reliability of the framework

In this thesis we tried to discover different concepts that are important to improve the alignment of requirements and design during global software development. As we have seen, this is quite a complex issue and has many alignment concepts related to this. We identified fifteen important quality attributes and this list is probably not complete. The most important concepts have been embedded in my framework.

Four in depth case studies were performed. More case studies are needed to improve the robustness of the chosen quality attributes and perhaps delete or add new important quality attributes for alignment. In addition, we still have to prove the validity of the framework in a prospective study, as we have only looked at past projects.

The ARAD framework only has been validated using RUP-based projects. The framework might not contain the same quality attributes when another development model is used that contains different requirements and design documents. In addition, it is not clear if the framework can be used in other organizations. Some artifacts can be used in a different way, which might mean that particular issues are overlooked in the current alignment framework. The external validity of the alignment framework can be achieved by a generalization of the ARAD framework.

7.4.2 Future users of alignment framework

The definitions presented in this research provide a common understanding about the issues that have to do with alignment of requirements and design. Every research project needs to do define definitions in order to reduce ambiguity. When talking about new quality attributes, there should always be a description and a goal in order to be sure what we are talking about and what we want to achieve. Potential future users of my framework should keep this in mind.

7.4.3 Research method and experiences

The research method implicated different important alignment concept from theory, as well as gathering information by means of interviews for different case studies. The followed approach was especially useful for the case studies performed and was also useful to get the theoretical part done. The information and knowledge gained from the case study interviews was comprehensive and gave me not only a better understanding about the problem, but also solutions about how to tackle these problems.

In the four case studies, the respondents only reflect their own opinion based on their experiences and this might contain opinions that are not true in general. Due to the availability of sources and time constraints it was not possible to interview more respondents for the case studies. In addition, during July, most people were unavailable due to the summer holidays or extremely busy with pending projects. Still, most information retrieved from the interviews could be used to see whether the alignment attributes were correct and resulted in a comprehensive picture about what things went wrong and what could be learned from these mistakes. This actually shows that the attributes chosen are important and shows the internal validity of the alignment framework.

The case studies chosen were of different size to see any differences in project complexity. One decision that relates to the size and complexity of a project is to have an architect in India. For big projects, like case study Alfa and Bravo, an architect in India might be very useful, as this will probably increase the understanding of the developers in India regarding the technical design. For small projects, the tradeoff is different, as the application is less complex and the cost of additional architect will be higher than the gains. Function points (FP) can be used to indicate whether the system is large (Pfleeger, S.L. 1998). Systems with more than 1500 FP were considered big in this research.

7.5 FUTURE RESEARCH

There are many things that can be explored in this research area. Moreover, the interviews and discussions with experts gave me some suggestions about new research issues that are very interesting. This paragraph will discuss four of them.

7.5.1 Mechanisms to share architectural knowledge

Sharing knowledge of software architectures is difficult, especially during global software development. Many problems may arise when this knowledge is not explicitly stored or communicated. Now that offshoring and virtual organization are becoming more and more popular, it is imperative to effectively manage this knowledge sharing process. As indicated by (Farenhorst, R. 2006), there should be sharing mechanisms tailored for this architecting process. It would be very interesting to look how these mechanisms can be constructed and what the impact will be of doing so.

7.5.2 Validating the framework in future projects

This research project has only looked at past projects. Once the guidelines defined in this study are embedded into the RUP model at the Result Centre of LogicaCMG, it would be interesting to look how future project cope with the same problems during global software development. The alignment framework should be applied in real projects along several years. It would be interesting to look at the validity of the framework in such a prospective study and to see if the framework and the related guidelines really produce significant improvements during global software development.

7.5.3 Defining metrics for important alignment concepts

This research used qualitative research methods to find out that there are fifteen major quality attributes that can improve or worsen the alignment of requirements and design during global software development. The findings in this research will be even more robust if quantitative techniques also acknowledge these quality attributes. In order to be able to use qualitative research, metrics have to be defined for each quality attribute for alignment. Furthermore, defining metrics for the quality attributes makes it possible to measure how well a project is doing by looking at the different values of the quality attributes.

For example: measuring the number of times the different artifacts are reviewed (with a tester, a builder and the customer) will indicate the consistency of the requirements documents. Ultimately, tooling for alignment can be made that indicates how well a project is doing and supports a development team to keep on the right track.

7.5.4 Alignment framework in other situations

As already stated in the previous paragraph, the alignment framework has been developed especially for RUP-based project at LogicaCMG and the artifacts they use. It is unclear what the differences are in terms of quality attributes, when looking at other development models. The kind of development model will influence the kind of requirements documents that are used, how requirements are managed, how people communicate and so on. Looking at different development models will probably give use even a better understanding about common alignment concepts that can be used to make a more general ARAD framework. Another strategy is to make a decision model that makes it possible for a company to quickly see which alignment model you should use during global software development.

The ARAD framework can be tested in other companies using RUP to see whether there are different important alignment concepts. As the RUP artifacts can be used differently, the quality attributes of the ARAD framework and their importance might also be different. Making a mapping of the kind of organizations and the different quality attributes for alignment will be very interesting matter to investigate. If this is known, a choice can be made what quality attributes are important for the different types of organizations. Besides that, it is important how the global development process is set up in a company. The different kinds of global development approaches probably have a few different alignment concepts that are relevant during an outsourcing project.

7.6 Final Words

The journey to discover what is important during alignment of requirements and design is far from complete. Yet, for me this exploration has ended. This research has focused on the communication dimensions during global software development. An in depth investigation resulted in various important alignment concepts that are important during alignment of requirements and design, especially during global software development and possible solutions were found to cope with the problems that are related to this. As we have seen, there are many topics that still have to be investigated. The previous paragraph describes a subset of the issues that can be explored. Of course, there are many more interesting topics for future research. The ones described in this thesis are closely related to my research and are therefore treated in the previous paragraph. Finally, I wish everybody success on all future endeavors in this research area, both in academics and in life.

Bon voyage!

REFERENCES

Adler, R. B. and G. Rodman (1988). Human communication: What and why? In: Understandinghuman communication. New York/London, Holt, Rinehart & Winston Inc.**:** pp.2-19.

Adya, M. P. (2006). "Imparting Global Software Development Experience via an IT Project Management Course: Critical Success Factors." Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06) **Vol. 1**: pp. 51-52.

Alexander, I. and S. Robertson (2004). "Understanding Project Sociology by Modeling Stakeholders." Software, IEEE **Vol. 21**(1): pp. 23-27.

Alves, C. and A. Finkelstein (2002). "Challenges in COTS decision-making: a goal-driven requirements engineering perspective." Proceedings of the 14th international conference on Software engineering and knowledge engineering: pp. 789-794.

Barbacci, M. R., et al. (1995). "Quality Attributes." Report CMU/SEI-95-TR-021, Software Engineering Institute.

Basili, V. R. (1992). "Software Modeling and Measurement: The Goal Question Metric Paradigm." CS-TR-2956, UMIACS-TR-92-96: University of Maryland.

Bass, L., et al. (2003). "Software Architecture in Practice", 2nd edition, Addison-Wesley.

Blaauboer, F., et al. (2007). "Deciding to Adopt Requirements Traceability in Practice." In: Proceedings 19th International Conference on Advanced Information Systems Engineering (CAiSE'07): pp. 294-308.

Bosch, J. and P. Molin (1999). Software Architecture Design: Evaluation and Transformation. Engineering of Computer-Based Systems, 1999. Proceedings. ECBS '99. IEEE Conference and Workshop**:** pp. 4-10.

Buschman, F., et al. (2007). "Pattern-oriented Software Architecture - A Pattern Language for Distributed Computing Volume 4", Wiley & Sons

Cin, M. D. (2000). "Structured Language for Specifications of Quantitative Requirements." High Assurance Systems Engineering, 2000, Fifth IEEE International Symposim on. HASE 2000: pp. 221-227.

Clarke, S., et al. (1999). "Subject-oriented design: towards improved alignment of requirements, design and code." Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications: pp. 325-339.

Clements, P. (2005). "Comparing the SEI's Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000." CMU/SEI-2005-TN-017.

CMMI Product Team (2006). "CMMI® for Development, Version 1.2 - Improving processes for better products." Software Engineering Institute, Pittsburgh.

Constantine, L. L. and L. A. D. Lockwood (2001). Structure and Style in Use Cases for User Interfaces. In M. van Harmelan, Ed., Object Modeling and User Interface Design. Boston, Addison Wesley.

Curtis, B., et al. (1988). "A Field study of the Software Design Process For Large Systems." Communications of the ACM **Vol. 31**(11): pp. 1268-1287

Damian, D. and D. Zowghi (2003). "An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations." Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) **Vol. 1**: 19.3.

Damian, D., et al. (2004). "An industrial case study of immediate benefits of requirements engineering process improvement at the Australian Center for Unisys Software."

Eck, P. A. T., et al. (2002). "A Conceptual Framework for Architecture Alignment Guidelines." project GRAAL WP1 whitepaper

Eeles, P. (2005). "Capturing Architectural Requirements."

Farenhorst, R. (2006). "Tailoring knowledge sharing to the architecting process", ACM Press. **Vol. 31:** art. 3.

Farooqui, K., et al. (1996). The ISO Reference Model for Open Distributed Processing - An Introduction. Computer Networks and ISDN Systems, Elsevier Science Publishers B.V. **Vol. 27:** 1215-1229.

Fowler, M. (1997). Analysis Patterns: Reusable Object Models, Addison-Wesley.

Gamma, E., et al. (1995). "Design patterns: elements of reusable object-oriented software". Boston, MA, Addison-Wesley Longman Publishing Co., Inc.

Gilb, T. (1997). "Quantifying The Qualitative: How to Avoid Vague Requirements by Clear Specification Language." Requirenautics Quarterly **Vol. 12**: pp. 9-13.

Gilb, T. (2005). Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage., Elsevier.

Gotel, O. and A. Finkelstein (1994). "An Analysis of the Requirements Traceability Problem." In: Requirements Engineering, 1994., Proceedings of the First International Conference on: pp. 94-101.

Grady, R. (1992). "Practical Software Metrics for Project Management and Process Improvement."

Gross, D. and E. Yu (2001). "From Non-Functional Requirements to Design through Patterns." Requirements Engineering **Vol. 6**(1): pp. 18-36.

Guetzkow, H. (1965). "Communications in organizations". In J. March (Ed.), Handbook of organizations. Chicago, IL, Rand McNally & Company.**:** pp. 534-573.

Guha, R., et al. (2004). "Contexts for the semantic web." In Proceedings of the 3rd International Semantic Web Conference **Vol. 3298**: pp 32-46.

Harsu, M. (2003). "From architectural requirements to architectural design."

Herbsleb, D. and A. Mockus (2003). "An empirical study of speed and communication in globally distributed software development." IEEE Transactions on Software Engineering **Vol. 29** (6): pp. 481-494.

Hevner, A. R., et al. (2004). "Design Science in Information Systems Research." MIS Quarterly **Vol. 28**(1).

Hofstede, G. (1994). "Cultures and organizations : software of the mind", McGraw-Hill.

Hofstede, G. (last accessed June 21st 2007). "Website Geert Hofstede Cultural Dimensions, http://www.geert-hofstede.com/."

Hoordijk, W. and R. J. Wieringa (2005). "Surveying the factors that influence maintainability: research design." In: Proceedings of the 10th European software engineering conference: pp. 385-388.

Hoorn, J. F., et al. (2007). "Requirements change: Fears dictate the must haves; desires the won't haves." **Vol. 80**(3): pp. 328-355.

IBM (2005). "IBM Rational Method Composer Version 7.0.1."

IEEE-Std.610.12 (1990). "IEEE Standard Glossary of Software Engineering Terminology/IEEE Std."

IEEE-Std.830 (1998). "IEEE recommended practice for software requirements specifications."

ISO/IEC 9126-1 (2001). "Software engineering -- Product quality -- Part 1: Quality model."

Jackson, M. (2001). Problem Frames - Analysing and structuring software development problems, Addison-Wesley Longman Publishing Co., Inc.

Jacobson, I. (2005). "Aspect-Oriented Software Development with Use Cases", Addison-Wesley.

Jacobson, I., et al. (1994). The OOSE method: a use—case-driven approach. Object development methods, SIGS Publications, Inc.**:** pp. 247-270.

Korn, J. (2001). "Design and Delivery of Information." European Journal of Information Systems **Vol. 10**(1): pp. 41-54

Kosman, R. J. (1997). "A two-step methodology to reduce requirement defects." Annals of Software Engineering **Vol. 3**(1): pp. 477-494.

Kruchten, P. (2003). "The Rational Unified Process: An Introduction", Addison-Wesley Longman Publishing Co., Inc.**:** 320.

Kruchten, P. B. (1995). "The 4+1 View Model of architecture." IEEE Software **Vol. 12** (6): pp. 42-50

Lasater, C. G. (2006). "Design Patterns" (Wordware Applications Library), Wordware Publishing Inc.

Lauesen, S. (2002). "Software Requirements Styles and Techniques", Addison-Wesley.

Lauesen, S. and O. Vinter (2001). "Preventing Requirement Defects: An Experiment in Process Improvement." Requirements Engineering Journal **Vol. 6** (1).

Lee, S. W. and D. C. Rine (2004). "Missing requirements and relationship discovery through proxy viewpoints model." Proceedings of the 2004 ACM symposium on Applied computing: pp. 1513-1518

Leffingwell, D. (2002). "Agile Requirements Methods."

Lewin, K. (1947). "Frontiers in Group Dynamics." Human Relations **Vol. 1**(2).

LogicaCMG (2007). "Visie document Ontwikkelstraten."

LogicaCMG, W. (last accessed May 27th 2007). "Blended Sourcing and Applications Management, http://www.logicacmg.com/corporate/350233331."

Majchrzak, A., et al. (2005). "Perceived Individual Collaboration Know-How Development Through Information Technology-Enabled Contextualization: Evidence from Distributed Teams." Information Systems Research **Vol. 16**(1): pp. 9-27.

March, S. T. and G. Smith (1995). "Design an Natural Science Research on Information Technology." Decisions Support Systems **Vol 15**(4): pp. 251-266.

Markus, M. L., et al. (2002). "A design theory for systems that support emergent knowledge processes." MIS Quarterly **Vol. 26**(3).

McCall, J. A. and M. T. Matsumo (1980). "Software Quality Metrics Enhancements." General Electric Company/Rome Air Development Center Final Technical Report RADC-TR-80-109 **Vol. 1**.

Morris, C. H. (1938). Foundation of the theory of signs. In International Encyclopedia of Unified Science. Chicago, University of Chicago Press. **Vol. 2**.

Nuseibeh, B. and S. Easterbrook (2000). "Requirements engineering: a roadmap." Proceedings of the Conference on The Future of Software Engineering: pp. 35-46.

Pfleeger, S. L. (1998). "Software Engineering – Theory and Practice", 2nd Edition, Prentice Hall.

Pfleeger, S. L., et al. (1994). "Evaluating software engineering standards." **Vol. 27**(9): pp. 71-79.

Radhakrishnan, R. (2004). "IT Infrastructure Architecture Building Blocks." Sun Professional Services.

Rumbaugh, J., et al. (1999). "The Unified Modeling Language Reference Manual", Addison-Wesley.

Shannon, C. E. and W. Weaver (1949). "The Mathematical Theory of Communication", Urbana, Illinois, University of Illinois Press.

Simmons, E. (2001). "Quantifying Quality Requirements Using Planguage", presented at Quality Week 2002.

Simon, H. A. (1996). "The Sciences of the Artificial (3rd ed.)." MIT Press, Cambridge MA.

Simons, J. H. A. and I. Graham (1999). "30 Things That Go Wrong In Object Modelling With UML1.3."

Soni, D., et al. (1995). Software Architecture in Industrial Applications. International Conference on Software Engineering, Proceedings of the 17th international conference on Software engineering. Seattle, Washington, United States, ACM Press**:** pp. 196-207.

Steehouder, M., et al. (1999). "Leren communiceren: Handboek voor mondelinge en schriftelijke communicatie", 4th edition. Groningen, Wolters-Noordhoff.

Tausworthe, R. C. (1980). "The work breakdown structure in software project management."

The Standish Group, I. (2001). "Extreme Chaos."

Tuckman, B. (1965). "Developmental sequence in small groups." Psychological Bulletin **Vol. 63**: pp. 384-399.

Verschuren, P. and H. Doorewaard (2000). "Het ontwerpen van een onderzoek", Boom/Lemma.

Weick, K. E. and D. K. Meander (1993). "Sensemaking and group support systems." In L.M. Jessup and J.S. Valacich (eds.), Group Support Systems: pp. 230-252.

Wieringa, R. J. and H. Heerkens (2004). "Evaluating the Structure of Research Papers: A Case Study." In: Second International Workshop in Comparative Evaluation of Requirements Engineering (CERE'04): pp. 41-50.

Wieringa, R. J. and H. Heerkens (2006). "The methodological soundness of requirements engineering papers: a conceptual framework and two case studies." Requirements engineering **Vol. 11**: pp. 295-307.

Yin, R. K. (2003). "Case study research: design and methods", Thousand Oaks, Sage Publications Inc., 3rd Edition**:** 181.

Zelkowitz, M. V. and D. R. Wallace (1998). "Experimental Models for Validating Technology." Computer **Vol. 31**(5): pp. 23-31

APPENDICES

APPENDIX A: ROLES & ARTIFACTS

This appendix explains the analysts and designer roles. Moreover, the important artifacts of the analysts and designers will be explained and the related roles will be listed.

The **analyst** role set consists of the following roles (IBM 2005):

- Business Architect; responsible for the overall Business Architecture, being involved in all the significant decisions regarding structure, main behavior and its realization, interfaces, constraints and trade-offs.
- Business Designer; details the specification of a part of the organization.
- Business-Process Analyst; leads and coordinates business requirements elicitation by outlining and delimiting the organization being modeled.
- Requirements Specifier; specifies and maintains the detailed system requirements.
- Stakeholder; an interest group whose needs must be satisfied by the project. It may be played by anyone who is materially affected by the outcome of the project.
- System Analyst; leads and coordinates requirements elicitation by outlining the system's functionality and delimiting the system.

The **designer** role set consists of the following roles (IBM 2005):

- Capsule Designer; ensuring that the system can respond to events in a timely manner, in accordance with concurrency requirements.
- Database Designer; leads the design of the persistent data storage structure to be used by the system
- Designer; leads the design of a part of the system, within the constraints of the requirements, architecture, and development process for the project
- Implementer; develops software components and performs developer testing for integration into larger subsystems, in accordance with the project's adopted standards.
- Integrator; leads the planning and execution of implementation elements and integration to produce builds.
- Software Architect; leads the development of the system's software architecture, which includes promoting and creating support for the key technical decisions that constrain the overall design and implementation for the project.
- User-Interface Designer; is concerned with user-interface design and means gathering usability requirements and prototyping candidate user-interface designs to meet those requirements.

For the analyst there are the several artifacts in RUP that are of importance. For each artifact the specific roles that are involved are specified. Table 24 shows the requirement artifacts that Analysts are interested in. Note that the roles involved are not only designer and analyst roles.

Table 23 - Important Analyst artifacts and related roles (IBM 2005)

Artifacts	**Roles involved**
Requirements consisting of:	
1. Glossary	System analyst
2. Requirements Attributes	System analyst
3. Requirements Management Plan	System analyst
4. Software Requirement	Software Architect Requirements Specifier
5. Software Requirements Specification	Requirements Specifier
6. Stakeholder Requests	System analyst
7. Storyboard	System analyst
8. Supplementary Specifications	System Analyst Business Designer
9. Use-Case Model	Designer System Analyst Business Designer
10. Vision	System Analyst

For the designer there are the several artifacts in RUP that are importance for making design decision. For each artifact the specific roles that are involved are specified. Table 24 shows the analysis and design artifacts that Developers are interested in. Note that the roles involved are not only designer and analyst roles.

Table 24 - Important Designer artifacts and related roles (IBM 2005)

Artifacts	**Roles involved**
Analysis and Design consisting of:	
1. Analysis Model	Software Architect Designer System Analyst Business Designer
2. Architectural Proof-of-Concept	Software Architect
3. Data Migration Specification	Database designer
4. Data Model	Database designer
5. Deployment Model	Designer Software Architect
6. Design Model	Designer

	Software Architect
7. Navigation Map	User-interface Designer
8. Reference Architecture	Software Architect
9. Service Model	Software Architect Designer
10. Software Architecture Document	Software Architect
11. User-Interface Prototype	User-interface Designer

APPENDIX B: ARTIFACT GUIDELINES

Most artifacts are specified, because is people are used to just make them, without keeping in mind that these artifact should have a purpose for the developers. As you can see in the previous paragraph, you can see that there are many artifacts looking only at the analysts and designers. However, it is recognized to only specify the views relevant for the system at hand (Bass, L. et al. 2003). The time wasted by specifying unneeded documents can be used for other purposes, like coding and testing. The important thing we need to keep in mind is what we really need to develop the system. Table 25 specifies certain guidelines that can be used (Leffingwell, D. 2002). Only if one or more criteria match, the artifacts are useful to design the system.

Table 25 - Artifact criteria (Leffingwell, D. 2002)

#	**Criteria**
1	*Communication of important system aspects* The document is needed for communication important system aspects or agreements, where simpler, verbal communication either poses to great a project risk or would be impractical.
2	*Team members work more efficiently* Documentation makes it possible for current and new team member to work more efficiently. New team members can speed up more quickly by the possibility to acquire important information specified in specific requirement artifacts.
3	*Future advantages* On the long term is has a real advantage to have the artifacts, as it will evolve and will be updated during the development, testing or maintenance activities of the system. Examples include use case and test case artifacts, which can be used again and again for regression testing of future releases.
4	*Imposed requirements* A company, customer ore regulatory standard might impose to specify certain requirements in an artifact.

APPENDIX C: CONTEXT CASE STUDIES

Case “**Alfa**”

#	Project characteristics
1	**Size of development team** 40 people, 20 in The Netherlands, 20 in India
2	**Time estimated and time needed in reality** Expected: 1¼ year In reality: 1½ year
3	**Development language** C#
4	**Function points (FP)** 2199

Case “**Bravo**”

#	Project characteristics
1	**Size of development team** For first subproject: 24 people (14 in The Netherlands, 10 in India) For the third subproject: 22 people (8 people in The Netherlands and 14 people in India) In the Netherlands: 3 analysts detached at customer, 1 analyst at LogicaCMG, 3 developers and 1 Project manager. In India: 8 developers and 6 testers.
2	**Time estimated and time needed in reality** For initial release: Expected: 7 months (For functional design 3 months, 4 month for building. First system was working after 1 year.) In reality: 1 year Two subprojects have already past. The project started 3 years ago and stopped for 1,5 year due to politics. Now development is resumed. This subproject started march 2007. By 2009, this project should be finished. There is already a delay, as the release of the previous subproject needed to be bug-fixed.
3	**Development language** C#
4	**Function points (FP)** 2250 FP for first subproject. The third and currently active project uses work breakdown structure (WBS),

	which is a fundamental project management technique for defining and organizing the total scope of the project, using a hierarchical tree structure (Tausworthe, R.C. 1980).

Case **"Charlie"**

#	**Project characteristics**
1	**Size of development team** 10 people (1 Project Manager, 1 system analyst, 1 tester, 1 architect, 1 developer in The Netherlands, 5 developers in India)
2	**Time estimated and time needed in reality** Estimated: August 2006 - June 2007 Reality: August 2006 - June 2007
3	**Development language** Java
4	**Function points (FP)** 230

Case **"Delta"**

#	**Project characteristics**
1	**Size of development team** 25 people (2 analysts, 2 designers, 1 architect and 5 developers in The Netherlands. 15 developers in India.)
2	**Time estimated and time needed in reality** Estimated: April – November 2004 Reality: April 2004 – February 2005
3	**Development language** Java
4	**Function points (FP)** 1200

APPENDIX D: INTERVIEW PROTOCOL

Overview of interview protocol

1. Introduction of the researcher

2. Social talk

3. Subject and goal of the interview

4. Explain that the results will be dealt with anonymously

5. State that the length of the interview is approximately 1 hour and 15 minutes

6. Structure of interview questions

Syntax Dimension questions

- Were the artifacts understandable?
- What artifacts were used during the specific project?
- What artifacts were missing for design?
- When to ship you design to the development location (at what point in the development life cycle)?
- What are the components reused for the project?
- How to improve the reuse of components?
- Were there any problems in design due to inconsistent information?
- How was this problem tackled and how should you tackle this problem in the future?
- What design patterns were used?
- What were the advantages of using the pattern?
- Who are the gatekeepers during the project?
- What information and knowledge is difficult to manage by the gatekeeper?

Semantic Dimension questions

- What domain knowledge and information is important to specify in requirements in order to prevent ambiguity?
- How to deal with ambiguity?
- Which artifacts did originally contain approximations errors?
- What approximations are difficult to make?
- Was certain environment context missing during the design of the system?
- What aspects about the environmental context are needed, especially when part of the design is outsourced to India?

Pragmatic Dimension questions

- Why were certain requirements not reflected in the initial design?

- What experience is relevant to have in your development team?
- What system requirements change a lot during design?
- How did the development team communicate the changed requirements?
- Was it difficult to communicate a requirement to a person of the development team in another geographical area?
- Did the development in geographical dispersed areas result in less dynamics of the team and why did that happen?
- What pragmatic aspects in communication of requirements to design went wrong due to cultural differences?

7. Conclusion

- What quality attributes are missing in the alignment framework?
- Thank the interviewee for his time and provide the planning for report of the interview.

APPENDIX E: INTERVIEWS

CASE STUDY "ALFA"

Report Interview Respondent A

Respondent	**A**	Interview date	**26-04-2007**
Role during the project	**System Analyst (requirement specifier)**		
Country where development took place	**Analysis in The Netherlands, development in India**		
Background/education	**Technological Innovation Sciences**		
Number of development projects involved	**15**		
Other important comments	**As this was my first pilot interview, not all questions in the template were asked. So at certain points you see a "-", which means the question was not asked or the respondent could not answer the question.**		

Syntax Dimension

Minimum requirements documents needed for design

The artifact were understandable and the artifacts needed are UC-model, UC-descriptions (90 descriptions), Storyboard, Vision & Scope, Non-functionals / Supplemental specifications, Glossary, Mapping of Screens & data, SAD, Use Case Realizations

Level of design needed in development location

Detailed design artifacts are needed at the development location like Use Case Realisations, Data Model and Sequence diagrams.

Reuse of building blocks

-

Consistent Information

There were style sheet problems, certain fields of the user interface were not consistently specified. Information concerning the graphical user interface, like what fields to use where and when to ask to save information. Also information to keep browser history or not. One use case contained an aspect that should have been specified in all use cases. Because the aspect was not specified in the rest of the use cases, it was neglected. Consistency was checked by:

- Internal check by tester and builder.
- External check by customer.

Consistency for user interface can be solved by making sure with the customer what fields should be implemented in the user interface. For the use case, this was added after we found out about the forgotten aspect. This was communicated by sending new documents concerning the change to India.

Analysis and Design Patterns

-

Gatekeeper information and knowledge management

There are two main gatekeepers, which are the System Analyst and the Lead-designer. For the System Analyst, information concerning changed requirements and all information from the other analysts in general is passed on to developers. For the Lead-designer information concerning patterns for design and design information from the architects is important to pass on to developers. For India contextual and consistent information is relevant, as the CMMI-level requires that a lot of things are explicit. They mostly only do what is specified, so you have to make sure most details are specified. Knowledge of past projects (experience) is important, as well as how to reuse components.

Semantic Dimension

Reduce Ambiguity

Only a few artifacts, mainly use cases, resulted in a specification that was ambiguous. There were around 10 things specified in an ambiguous way. The customer did not get it exactly as they desired.

Defining quality requirements

-

Enough context of the surrounding

Authorization was not always clear. Furthermore, not all interfaces with other systems were specified, like the communication with the VGZ-system.

Pragmatic Dimension

Verification of requirements

Due to time constraints certain requirements were not reflected in the design. Due to these constraints around 80% of the requirements were implemented in the first design.

Required skills and experience in development team

-

Changing requirements

Style guide and supplemental specification change a lot. Steps done in business process modeling (BPM) also change. The kind of data used changed. Most of the changed requirements were implemented, as there is generally a strong focus to implement the requirement that was changed. Due to time constraints less than 10 % was not implemented during this project.

Availability team members

In India there are a lot of holidays, which means a lot of people are not available. This means there is a delay in developments at certain stages. The designers at India retrieve the requirement of artifacts by means of e-mail.

Team dynamics

-

Cultural differences

For Indian people it is more important that they stay on they high level of CMMI. The mistakes are sometimes not addressed to the right category for this reason. People do not really stand up for themselves or introduce new ideas. If they know a better solution they will not tell you, as they do not want to act as a dominant member of the team. It is more important to just get the work done, even if their idea might be better on the long run.

Report Interview Respondent B

Respondent	**B**	Interview date	**08-06-2007**
Role during the project	**Lead Architect**		
Country where development took place	**India**		
Background/education	**HEAO & business informatics**		
Number of development projects involved	**10+**		
Other important comments	**-**		

Syntax Dimension

Minimum requirements documents needed for design

Not all artifacts were understandable, as certain use cases contained open questions and were not processed completely. Todo's in artefacts did occur regularly, which means the artifact was not completely specified. There were some problems with people doing only their specific part. Some requirements specifiers did not want to do other things than specify use cases. I really needed them to do more and be a team as a whole and support each other.

The artifact needed are UC-model, UC-descriptions (90 descriptions), Storyboard, Vision & Scope, Non-functionals / Supplemental specifications, SAD, Glossary, Mapping of Screens & data and Use Case Realizations.

Level of design needed in development location

Detailed design is done in The Netherlands for this project. However this is not necessary, after analysis phase and when the data mode is specified, the technical design can be done in India. You have to specify in two languages, for the customer in Dutch and for the Indians in English. An excel sheet for terminology was ultimately made where entities are mapped to screens. So, after specifying enough contextual information in English, the design can be done in India in theory. Only data model needs to be specified thoroughly before shipping design. In practice, higher level-design ensures that Indians have a better understanding and prevents wrong design decisions. For non-standard projects contextual information is important. If the project is about a standard system, contextual information is not needed.

Reuse of building blocks

The building blocks used are the Microsoft framework and using code generation templates that makes use of the database. No real problems due to lack of knowledge, as the Microsoft framework is well known.

Consistent information

There was a problem that the requirement specifier did not specify realistic requirements. The customer likes this, as everything they want is promised, however this is not always possible. The artifact style guide was missing for design. Specific functions were specified in two different ways, so you could implement it in two fundamental different ways. The consistency was improved by specifying the style guide and dropping the not realistic requirements.

Analysis and Design Patterns

No real design pattern was chosen. Some patterns are already imbedded in Microsoft frameworks. Using the Microsoft framework made it possible to generate code using the database. Standard tooling was available using the framework. The framework also made it possible to have a good balance of the lead.

Gatekeeper information and knowledge management

No real gatekeepers during project. According to the respondent, there should also be an architect in India that can communicate the design to the developers. He is the linking pin to designers in India. He can clearly communicate technical constraints and how to make use of designers and available tooling. This makes direct communication possible and results in more interaction. Technical information and constraints are difficult to manage, as well as how to make use of people knowledge and to decide what tooling to use for development.

Semantic Dimension

Reduce ambiguity

Domain knowledge was pretty good, because a glossary was being made with clear definition of words. All relevant nouns were specified in the glossary. Making use of the glossary reduces ambiguity.

Defining quality requirements

There were performance issues, because no metrics were used to define non-functional requirements. It was not clear how many record a table would contain. This was significantly more than initially thought. Non-functional requirements were not specified in artifacts and an assumption was made of the amount of records.

Performance is difficult to define and the information relating to these non-functional requirements was lacking. The architect could not make a good estimation of performance. Al lot of technical issues were difficult to estimate. The customer also wanted a google-like search function on each screen. It was unclear what information should be searched for, which resulted in a highly inefficient search function. Normally they search only for specific information relating to the field of the screen. The requirement specifier did not ask what the customer really wanted to search for.

Enough context of the surrounding

External links were specified good enough. Technical it was clear, however specific functional design information had lack of environmental context. The process at the customer was specified with not enough details and it was unclear how everything really worked. Examples are when to do a payment and what checks need to be included. Enough functional details should be specified in the requirement artifacts.

Pragmatic Dimension

Verification of requirements

The interviewee does not know why. Mark Artz knows.

Required skills and experience in development team

For analysts, it is important he has experience in the same market segment and know what they are talking about. During this project there was an analyst that was already familiar with insurances, which really helped to know how the processes of the customer work.

Changing requirements

Certain requirements change, as certain processes were not clearly described by the customer. How to calculate a insurance costs. This was changed several times.

There were 5 kinds of declarations, which turned out to be only four. This means that one of the declarations was something else and needed to be changed afterwards.

Availability team members

In India they start 3,5 hours earlier. They were always available for us, so during this project availability of Indians was no issue. By means of telephone and chat sessions requirements were communicated.

Group dynamics

People implementing similar functionality should stay developing at that specific part of the system. They know the details of the domain and know already the people they are working with. This approach gives no new start-up problems.

Cultural differences

Sometimes the Dutch are too blunt for Indians and they might feel offended. Communication training helps to understand the Indian people. Each person reacts differently and it is difficult to have one approach to deal with cultural problems.

Report Interview Respondent C

Respondent	**C**	Interview date	**23-07-2007**
Role during the project	**Project manager India**		
Country where development took place	**India**		
Background/education	**MSc. Civil**		
Number of development projects involved	**10**		
Other important comments	**-**		

Syntax Dimension

Minimum requirements documents needed for design

Yes. The mainly the artefacts for the design are use cases. Mainly due to translation some issues were missing.

Level of design needed in development location

High level design is done in NL and low level is done here.

Reuse of building blocks

The components reused are data access layer components, user controls and business classes. Come up with documents and update them frequently when there are changes to the classes.

Consistent information

There were problems and most of them will be found at the design review. Implementing CMMI standards to project will help to have consistent information.

Analysis and Design Patterns

Microsoft design patterns like Application data block were used that have proven to work in previous projects.

Gatekeeper information and knowledge management

-

Semantic Dimension

Reduce ambiguity

From DGVP project point of view there is no specific business domain. Understanding the customer workflow and putting it in the document is the important part. Doing reviews helps to reduce ambiguity.

Defining quality requirements

-

Enough context of the surrounding

No, not missing context. The important data is the customer environment configuration so the system testing people can test the application in the same environment.

Pragmatic Dimension

Verification of requirements

Requirements were not reflected in design due to lack of reviews.

Required skills and experience in development team

Experience in the RUP phases is a must and understanding of CMMI processes is preferred.

Changing requirements

Requirements related to user interface of the system change a lot. Through change request requirements were communicated.

Availability team members

The team is well connected to each other through mail, communicator and other means. Since the team work according to the plan the time difference won't have an effect on development project.

Group dynamics

-

Cultural differences

-

CASE STUDY "BRAVO"

Report Interview Respondent A

Respondent	**A**	Interview date	**30-05-2007**
Role during the project	**Project Manager first subproject**		
Country where development took place	**Analysis in The Netherlands, development in India**		
Background/education	**HTS – Electronic & Computer Science**		
Number of development projects involved	**15**		
Other important comments	**This was one of the first projects where development was done using the RUP development method. The project manager was running the first sub-project. The whole subproject is actually divided into 3 subprojects.**		

Syntax Dimension

Minimum requirements documents needed for design

Not all artefacts were expressive enough. The quality of use cases was not good enough. A lot of supplemental specifications were needed to make all things clear to developers. Artifacts needed are: Glossary, Storyboard, Supplemental specification, UC-model, UC-descriptions (90 descriptions), Vision & Scope, Analysis model, Architectural Proof of Concept, Data model, Requirements Management Plan, Software Requirements, Deployment model, Design model, Navigation map.

Level of design needed in development location

Technical design documents are really important, as this decreases the risk of different interpretations. Details are specified to ensure the developers know what to build. It happened that documents were read and interpreted totally different than intended.

Reuse of building blocks

Certain building blocks from Microsoft were used. This was a framework that could be used for our project. There was training about what to do and to gain knowledge about C#, Visual studio and 3rd party tools. So, this did not really cause any problems

There was one code base that resulted in direct identification of problems, as changes in code were directly noticed. Potential problems were tackled on the fly.

Consistent information

Yes, there was a problem with terminology. For this reason a glossary in Dutch, as well as English was send to the translator to make things as clear as possible. This ensured everybody understood the documentation. In certain use cases, not all terms were interpreted the same, which created use cases that did not match. So, there was a problem of no consistent information in use cases. Checks were done by a peer review, a customer and testers for test plans. Review and feedback of use cases and better clarification of the use cases help to improve consistency. The following artifacts needed to be checked: Use cases, unclear, does it really explain how things should be done. Storyboard, making a screen satisfying needs of customer and looking if this is possible with developers. Design model, does the design explain the system to be developed clear enough.

Analysis and Design Patterns

The respondent does not know.

Gatekeeper

Not really a gatekeeper, for the functional design there was one requirement specifier available to answer questions about use cases. Functional documentation is difficult to manage, as this is open to own interpretation and difficult to communicate to developers.

Semantic Dimension

Reduce ambiguity

Use cases were very unclear sometimes. Furthermore requirement attributes were not very precise. When using supplemental specifications to make use cases more understandable, there was sometimes still ambiguity. So these supplemental specifications were also too ambiguous every now and then. As a final point documents describing abstract level things, like Vision & Scope were not defined in a very clear way, which resulted in misunderstandings sometimes. To have enough domain level knowledge more details in use cases is needed. Also use appropriate words when talking at abstract level and not words that do not say anything.

Defining quality requirements

None, there already was an architecture. The proof of concept already existed, so defining quality requirement was not required.

Enough context of the surrounding

In the beginning phase there was a lack of knowledge about how the system should work and why. This was explained by means of functional documentation and direct communication. Functional design is important for India to have a clear explanation why the system is developed. In addition, technical design is important to communicate the kind of database used and coding guidelines/naming conventions.

Pragmatic Dimension

Verification of requirements

All requirements specified were implemented during design. However during the testing phase we noticed certain requirements were not implemented, which means problems in the later phase of software development of RUP.

Required skills and experience in development team

Experience with RUP models and how to use it. RUP experience was not satisfactory at that moment, as this was one of the first projects using the RUP method at LogicaCMG. Learning how to read the models was difficult, as there was no experience using the new models in RUP.

Making people aware of the development model and the disciplines and their role during the project was really important.

Changing requirements

Requirements relating to the user interface changed a lot. After a change and making a new use case for example, the building normally started immediately. Changes were mostly done in The Netherland, because it took to much time to explain everything. By the time everything was explained in India, it would already be implemented in The Netherlands.

There are currently still new requests for change. This can be done very quickly, however there are two releases of the system per year, which could mean that the newly specified requirement is embedded in the new release after 6 months.

Availability team members

In October there is a low availability, due to the many holidays. Working during the weekend is not a problem in India. It happened that there was a holiday that we did not know about, but this did not really cause any problems. The time difference is 3,5 hours during the summer so they start 3,5 hours earlier, which is not really a problem. Normally the time difference is 4,5 hours.

Group dynamics

-

Cultural differences

During a project it is sometimes better that the project managers tells a developer what to do and not a colleague due to the hierarchical issues.

Report Interview Respondent B

Respondent	**B**	Interview date	**01-06-2007**
Role during the project	**Information analyst and coaching first subproject**		
Country where development took place	**India**		
Background/education	**Technological Innovation Sciences**		
Number of development projects involved	**12**		
Other important comments	**There are 3 different subprojects during this project; we only focus on one of them, the first project. Notice, the system is still changing, as new functions are added regularly.**		

Syntax Dimension

Minimum requirements documents needed for design

Translation of use cases was quite bad in the beginning and caused that additional explanation was needed. Artifacts needed during this project were Glossary, Storyboard, Supplemental specification, UC-model, UC-descriptions (90 descriptions), Vision & Scope, Analysis model, Architectural Proof of Concept, Data model.

Specific technical artifacts were missing at abstract level. Furthermore the quality of functional requirements artifacts was insufficient, as most people were unfamiliar with the RUP artifacts. Most initial use cases were translated by some external party that was lacking certain contextual things. After a while most artifacts were specified according to RUP method and were translated by the requirement specifier itself. This resulted in good quality artifacts, as all contextual aspect was clearly defined.

Level of design needed in development location

For first sub-project, functional design was done in the Netherland and technical design and implementation was done in India. This did not always work smoothly, as there were no real technical guidelines.

In theory proper functional design and an abstract technical design with guidelines is needed in the development location. This abstract level design can be done in 2 pages.

Reuse of building blocks

-

Consistent information

Specific words were not consistently used, which resulted in unclear specifications. Use cases contained several words describing the same thing; however the word could be interpreted differently.

It is important to choose only one word and no synonyms. These words were described in the glossary and really helped alignment of communication. Also personal relations are important to keep a healthy development team. Inviting Indian people to come over and learn how things are done in The Netherland helped them to get a better understanding about our development approaches. Use cases are most important to check. For requirement analyst review is done as follows: colleague does peer review, review of the customer, intake with architect, between these different reviews, there are feedback loops to adjust the use cases.

Design patterns

No software design patterns were used. This is why certain things went wrong, as there was little structure sometimes in building the application and tackling certain problems when using pattern that solve these problems.

An advantage would be that software has a better design and is easier to maintain for example. However this was not really done in this project.

Gatekeeper

There was not really a single point of contact. However, the analyst communicates the requirements (system) to developers. This was also my role in this project. The project manager manages deadlines and how and when to deliver amongst other things, but there was not really persons that always managed these information channels. Knowledge about patterns and how development is done with these patterns is difficult to manage. Communication was very difficult due to language barriers. A good solution is to require developers and requirements specialist to have adequate (English) language skills.

Semantic Dimension

Reduce ambiguity

For translation of the use cases a translation agency was used. This was not a success, as they used different kind of words for the same thing and did not provide enough contextual information for Indian people. Normally in the Netherlands this is not a problem, because there is direct communication between the developer and information analyst who will answer the questions the developer has. Making use of a glossary where relevant nouns are clearly defined can help to reduce ambiguity.

Defining quality requirements

During this project quality requirements were not specified in a document. A lot of estimates were made in an informal way and not written down. There was a lot of communication between the analyst and the architect. If design is done in India, these things should be written down in a document to prevent misunderstandings.

Enough context of the surrounding

The problem domain was not really clear for developers. At the start of the project Indian people did not always exactly know to customer and what the customer really wanted, like quality, speed of development and project standards and their contribution in the whole development life cycle.

There should be awareness of the project context. This means you have to know what the Indian developers do, who the customer really is, customer preferences regarding quality or speed for example and project standards that indicate how to the development is going to be done.

Pragmatic Dimension

Verification of requirements

Everything of the first requirements set was implemented and reflected in design. Sometimes a requirement was not implemented the way it was intended, so those parts of the system needed to be altered.

Required skills and experience in development team

Decent English is really required in your development team. Sometimes I could barely understand the Indians. During offshoring there is lack of experience with our current customers and their business processes. Indians do not have enough contextual information how things work in a company sometimes. To cope with this, they should first read about the company that wants to have the new system and understand their problem. After that they should read the requirements. This will make sure they have enough contextual information. This requires that most important contextual information is specified in an additional artifact. During a long lasting project the development team should be updated with current processes of the customer that maybe have changed, to ensure that their design is aligned with the demands of the customer.

Changing requirements

Changing requirements can have several reasons. New tasks may be added of the customer. There might also be politics involved when there are basically 2 groups that want to have different functions. If a certain group wins, their functions will be implemented. However, if the dominant group is changed, these functions have to be deleted or updated to the liking of the other group. There are 3 different kinds of requirements that can change:

- Process requirement (use cases)
- Data requirements
- Presentation requirements (GUI, Style guide)

LogicaCMG is does not have enough good Usability experts. This is why GUI aspects frequently change.

Availability team members

There are a lot of communication possibilities. However, communication media are not used frequently to communicate with Indian developers. Possible media that could be used are video conferencing and Net meeting to make drawing for example.

Group dynamics

The team cannot communicate directly, which makes the team less dynamic and causes interpretation problems.

Cultural differences

There was a “we” and “they” feeling. This feeling was stronger at the Dutch development team. The India people were seen as strangers.

Report Interview Respondent C

Respondent	**C**	Interview date	**08-06-2007**
Role during the project	**Project Manager third subproject in NL**		
Country where development took place	**India**		
Background/education	**Aerospace Engineering, Delft**		
Number of development projects involved	**10**		
Other important comments	**The third subproject is treated in during this interview. Hence that the project is still running.**		

Syntax Dimension

Minimum requirements needed for design

The artifacts were not really understandable. First, we tried to use the standard RUP artifacts. However, the customer did not understand. They wanted to use their own software requirement specification (SRS) that was basically a word-document. 3 analysts are currently detached to the customer, helping them with the requirement specification. The requirement specification that was initially used should never been accepted and contained already insufficient information in the Dutch version. Proper use cases and artefact specified by RUP are needed for design.

Level of design needed in India

The customer is does the functional requirements part themselves, with help of 3 analysts from LogicaCMG. Technical design is done in the Netherlands. During this project there might be time constraints that prevent a proper design to be made. Hence the project is still running

Reuse of building blocks

The previous two sub-projects contain building blocks that can be used for this last sub-project. Bringing Indian developers to The Netherlands can help to transfer tacit knowledge concerning the building blocks.

Consistent information

In the SRS there were inconsistencies that were noticed during an intake by comparing the artifacts.

Analysis and Design Patterns

-

Gatekeeper

There was one linking pin for the analysts, who communicated the functional documentation. For technical questions there was also one point of contact here in The Netherlands. There were not really gatekeepers in this project managing all information channels.

Information in "Clearquest" is difficult to manage and making a comprehensive picture is hard. Contextual information is important for Indians, Indians can come to the Netherland to get to know how we work and who the customer really is, including their processes.

Semantic Dimension

Reduce ambiguity

Certain domain information was communicated verbally a few times and ultimately documented. However, this resulted in certain mistakes, because the initial message was formulated different due to different interpretations. Specify the domain information directly and verify with customer after doing so may help to reduce ambiguity.

Defining quality requirements

Performance in supplemental artifact was not explicitly expressed in a metric, which resulted in approximation errors. Performance estimates of the system to be developed are difficult to make. There were not enough documents for the architect to make a good design decision. This resulted in performance problems, as the program was could not process all requests in an efficient way.

Enough context of the surrounding

-

Pragmatic Dimension

Verification of requirements

Specific demands of the customer were interpreted differently, which resulted in a design that did not contain this specific demand. Hence the third subproject is still running.

Required skills and experience in development team

People with different skills and experiences are desired. Furthermore, team members have to be able to listen to each other and not only look at a problem from only their point of view. This prevents unnecessary discussions. There was one person already familiar with the kind of business of the customer and this helped us to get a better understanding.

Changing requirements

Due to new laws, the system needed to be altered. Normally we send the new specifications to Indian people with enough contextual information. From previous project we experienced no real problem here. In India they do everything correct if you specify enough details.

Availability team members

There is a time gap of 3,5 hours. Sometimes Indian developers are not available, as they leave earlier. This time difference also has an advantage that we can alter the system at the end of the day, while they do not notice, because they are not working at that time. The next morning they start earlier than we do and this creates a more time efficient process.

Group dynamics

You do not see Indian developers and only talk to them by means of office communicator. There is regularly contact and there are also conference calls now and then. This is sometimes awkward, as it is difficult to understand the Indians. There is a good collaboration by means of chat sessions.

Cultural dimensions

If something, a requirement for example changes, Indian developers do not ask a lot of questions. Not all code is altered if there is a bug fix; they only fix the code at one place for example, while there are 4 places where this fix is needed.

Certain code is written not using our coding conventions. They think their solution is also a good one and do not ask if it is correct. Normally, in The Netherlands they will ask for approval to check if it really is ok.

Report Interview Respondent D

Respondent	**D**	Interview date	**14-06-2007**
Role during the project	**Project Manager India third subproject**		
Country where development took place	**India**		
Background/education	**MSc. Software Systems**		
Number of development projects involved	**1**		
Other important comments	**The third subproject is treated during this interview**		

Syntax Dimension

Minimum requirements needed for design

The artifacts for the design are mainly use cases. Some of the information was missing in the documents, since they were first prepared in Dutch and translated into English, "Lost in translation".

Level of design needed in development location

Indian team role starts when the design phase is over in The Netherland. Once the design was completed and agreed upon, it will be shipped to India. Here we will do a review of the design and make sure that it meets all requirements criteria. This is called the design sign off and coding phase starts.

Reuse of building blocks

The components reused are data access layer components, user controls and business classes. Business classes resulted in understanding problems due to lack of proper documentation.

Consistent information

There were problems and most of them are noticed during the design review. We implemented extensive design review and captured the review comments to analyze. We follow CMMI quality processes in India so the analysis of data (metrics) will help us to improve in future.

Analysis and Design Patterns

MS design patterns like Application data block and disconnected architecture. The patterns are already proven ones, which ensures the solution will work.

Gatekeeper information and knowledge capabilities

This can be answered by the Dutch team.

Semantic Dimension

Reduce ambiguity

From the project point of view there is no specific business domain. Understanding the customer workflow and putting it in the document is the important part. Doing reviews helps to deal with ambiguity.

Defining quality requirements

Estimates are difficult to make, when functionalities are not clear.

Enough context of the surrounding

Not really missing context of the surrounding. The important data is the customer environment configuration so the system testing people can test the application in the same environment.

Pragmatic Dimension

Verification of requirements

Requirements are mostly not reflected in design due to lack of reviews. Hence the project is still running, so cannot really answer this question.

Required skills and experience in development team

Experience in software development life cycle phases is a must and understanding of CMMI processes is preferred.

Availability team members

-

Cultural differences

None. The respondent does not know the cultural dimensions and differences of the Indian team and the Dutch team.

CASE STUDY "CHARLIE"

Report Interview Respondent A

Respondent	**A**	Interview date	**2-7-2007**
Role during the project	**Lead Architect**		
Country where development took place	**India**		
Background/education	**Technical Computer Science, Open University** **Air & Space Aeronautics Architect at Fokker**		
Number of development projects involved	**25 (5 at LogicaCMG)**		
Other important comments	**First ING project using RUP**		

Syntax Dimension

Minimum requirement artifacts needed for design

The artifacts were understandable, only the translation in English was sometimes incorrect. Artifacts needed are Use Case Model, Vision (start, context and demands), Supplementary specification (FURPS, ISO 9126), Glossary for India, Prototype important screens, Domain model and SAD.

Level of design needed in development location

For this project, design en development was shipped to India, after the most important components were implemented in The Netherlands. Only straightforward components that were less crucial for the system were designed and implemented in India.

Reuse of building blocks

Not many, a few login components were used from the bank itself. Also transformation for images and infrastructure were used. From Result, configuration application was reused.

Furthermore a standard Spring framework was used. The Spring framework uses inversion of control and code application containers to make it easy to maintain. All references were generated once and it increases runtime performance, while improving test coverage and application quality. With inversion of control, each function is seen as a separate bean that does not know about other classes of the system. The Spring container ensures that the bean has all services need for the bean to function properly. The spring container knows when a bean needs another bean. So the Spring container knows which bean is needed, where it is needed and which setter has to be used to make use of the bean.

JBPM was also used, which is an open source workflow management system. Furthermore, Hibernate was used, which is a powerful, high performance object/relational persistence and query service. Hibernate help you with relational persistence following object-oriented idiom. This includes association, inheritance, polymorphism, composition, and collections.

Indians had a knowledge problem, as they were not familiar with Spring. The developers in India were mainly doing plain java work.

During the project we tried to divide the work to people already having enough skill and experience to design and implement the components. So there basically was a mapping between work and knowledge of the developers. During a bigger project it would be very useful to have an architect in India that can communicate the technical design to the developers in India. This project was too small to justify an architect being sent to India

Consistent information

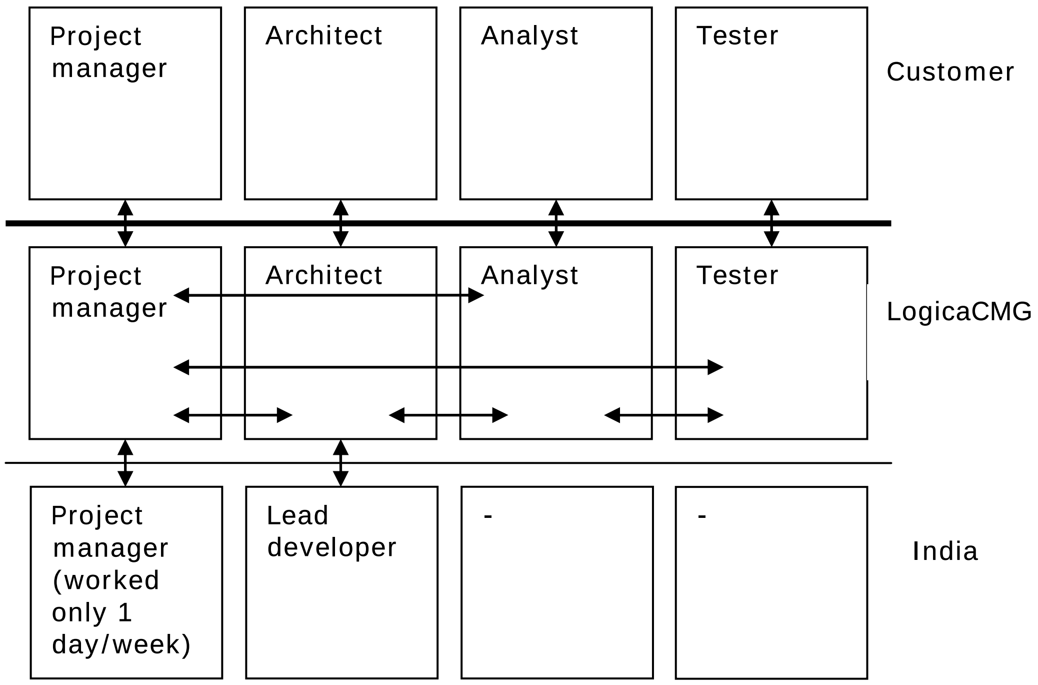
There were problems with the interfaces of the system. The first documentation contained no interfaces, so it was unclear to which external systems the system needs to connect. Furthermore there was no clear description of the behaviour of the system and during the design certain behaviour did not match. This information created inconsistency in the design, as it was unknown what events cause what actions. In India there were inconsistencies in code. Similar functions could easily be reused; however two Indians developers made totally different code, which is also hard to maintain. This also indicates that they did not communicate with each other, which could have saved them some time. The analyst should explicitly ask which interfaces and services should be used. During this project assumptions were made.

Analysis and Design Patterns

Used patterns are Model View Controller, Layered, Façade, Tranfer object, Singleton, Abstract factory. The Spring framework contained already certain patterns, like Model View Controller, Proxy, Singleton, Abstract factory and data access. The call of interfaces was really easy with the Spring framework. There was high flexibility of dependencies between objects, as dependencies were not hard coded. Interface and implementation used Oracle and Hibernate that uses XML configuration files. Hibernate can be used for generating tables for a database application. This includes adding data to the tables, retrieving, updating, and deleting table data.

Gatekeeper

One person, the lead developer in India was assigned to communicate the requirements and design to the other Indian developers. This was not very successful, as there were still some misunderstandings about how to implement certain functions. Actually there was no real gatekeeper during the project. There was lack of contextual information relating to the primary processes of the bank. Knowledge regarding the frameworks that are used can be understood by Indian developers, but in most cases there is no time to learn this. So, due to time constraints it is difficult to transfer the knowledge regarding technical design. For communication see the figure below, where each arrow indicates communication efforts.



Semantic Dimension

Reduce ambiguity

Information relating to available services of other systems that clearly describes the business logic is important to specify. Contextual information was difficult to transfer by means of MSN messenger, which we mostly used to communicate. Vision document is useful to get to know specific contextual information. An external agency translated the documentation, like Use Cases. This was not satisfactory, due to the bad translations. A glossary helps to deal with ambiguity.

Defining quality requirements

Supplemental specifications contained wrong estimations. These estimations had to do with efficiency of the system. The number of orders per day was incorrect. Actual number turned out to be twice as high. Averages are not always useful, as there might be a huge fluctuation in number of transactions during the day.

Enough context of the surrounding

It was difficult to get all contextual information, as communication with the customer with the customer was difficult. This is now solved, as there is good direct communication with the customer. Documents describing the interfaces of the system and a list of the services that are available is important in order to understand the context of the surrounding.

Pragmatic Dimension

Verification of requirements

Almost all requirements were reflected in the design. Only “making reports” function was out of scope and dropped by the customer due to time constraints.

Required skills and experience in developments team

Skills and experience related to Spring and data storage possibilities is important. The résumé of Indian developers was not always correct, as the skills and experience did not reflect the real situation. The lead developer for example had trouble with object orientated design, however his résumé indicated that he was familiar with this.

Changing requirements

The requirements relating to the interfaces changed. Sometimes there was and extra workflow steps in the transaction process. Changes were mainly done in The Netherlands and communicated directly. Afterwards documents were changed and updated, however sometimes this was forgotten.

Availability team members

There were no real problems relating to availability of team member in India. There are a lot of holidays in India, which can be a problem. However, using proper project planning does tackle this issue. It occurred once that there was a strike. The exact reason for this strike was unknown. When there were some problems that could be solved in India at the end of the day in The Netherlands, this was send to India. This ensures the problem is dealt with immediately the next day early in the morning.

Group dynamics

The role of the Lead Developer in India did not really work as wanted, as there were still misunderstandings in the Indian team. The reason for this is unclear. Perhaps the caste system causes the people higher in the hierarchy not to listen to people lower in the hierarchy. The Lead Developer has been in The Netherland during the inception phase and should have enough contextual information about the customers’ processes amongst other things. Reuse of code could have been better in India, as the developers could have reused each others code. Perhaps this indicates that there was a less dynamic group in India. The Lead Developer didn’t intervene either to improve communication efforts.

Cultural differences

To make remarks about the work of an Indian developer is dangerous, as they might feel offended. It occurred that one Indian developer stopped working, due to a blunt remark. They do not take everything you say for granted and can be stubborn sometimes. The Indian developers sometimes did not understand the Use Cases, but did not indicate this.

Report Interview Respondent B

Respondent	**B**	Interview date	**13-07-2007**
Role during the project	**System Analyst**		
Country where development took place	**India**		
Background/education	**HEAO & business informatics**		
Number of development projects involved	**10 (6 years)**		
Other important comments	**First project using customized RUP**		

Syntax Dimension

Minimum requirement artifacts needed for design

The artifacts were translated by a translation agency, which had no affinity with IT and this caused wrong translation of IT terminology. Artifacts used are Use Case Model, Vision (start, context and demands), Supplementary specification (FURPS, ISO 9126), Glossary, Prototype important screens, Domain model, SAD, Navigation Maps, Mapping screen and data (Necessary artifact), Activity diagram with swim lanes (to clearly see which activities take place where). The prototype was important for the actors.

Level of design needed in development location

Dutch developers do not need everything specified. However, in India a very detailed design is needed where nothing left open for own interpretation. The data model needs to be specified thoroughly. Furthermore, a prototype will enhance the understanding of the Indian developers to get to know how the system has to look like and is really important to prevent a wrong GUI being build.

Reuse of building blocks

-

Consistent information

Specific messages could not fit in a text field due to inconsistent information. The interfaces from ING were not clearly described by them. The simulation behind the interfaces was unclear. Supplemental specifications should be checked for inconsistencies. Check consistencies of artifacts by doing reviews of artifacts produced from the customer.

Analysis and Design Patterns

Ask the architect for design patterns. A functional pattern that was used is the party pattern. In this pattern, a Party is an abstraction to define persons or organizations. A Party class for example has sub-classes of Person and Organization. The Party could have different roles such as a person could be an employer or employee, a manager, etc. An organization could be a business entity, a shelter, a university etc.

Gatekeeper

Gatekeepers are the System analyst and Architect/Lead developer. A System analyst is the gatekeeper for the functional design and the Architect or Lead Developer is the gatekeeper for the technical design.

Semantic Dimension

Reduce ambiguity

Relationship between entities in domain model is difficult, but really important to specify in requirements specification. Using only one word for the same thing and do not use synonyms. Pictures of activity diagram with use cases would help to improve domain information. To categorize in must haves or not, is difficult. The customer might think that a function is important; however this might not be the case. Use Requisite Pro to manage and see traceability of requirements.

Defining quality requirements

-

Enough context of the surrounding

The stubs/interfaces we got from the customer were not completely specified. It was difficult to see the relationships between external systems. The level of security for the portal was difficult to determine. The kind of services available was not clearly described. Documentation of other systems from the customer also lacked.

Pragmatic Dimension

Verification of requirements

Monitoring, making reports and scanning of fax orders requirements were not reflected in the first design due to limited budget. They first wanted to have basic functionality of the system and later maybe add additional features.

Required skills and experience in development team

Important skills are listening, patience, communicative, flexible when looking at other requirements, pragmatic and analytic skills. Analytical skills are really important, as you have to decide whether the problem identified by the customer really is the problem they want to be solved. You have to be able to play the role of the user of the system. Technical background is important in order to know what is technically feasible or not and thus accept, alter or deny specific requirements from the customer. You have to be up-to-date with all new IT-developments.

Changing requirements

GUI requirements like for example that the text field was too small to fit the message. Interface changed, due to the fact that you could still use to old application to alter fax messages. Communication occurred directly.

Availability team members

There was an incident that the code did not match criteria. When the developer was asked to change the code, in our direct Dutch way, he refused. When we asked why, we found out that we didn't take in account that yesterday was an Indian holiday that is equal to our new year's day.

Communication occurred by means of chat application and sometimes telephone. Indians start earlier and sometimes the Dutch team could not "check in" and start working immediately, because the Indians were still busy.

There was a preference to use e-mail, as this makes sure the changes of requirements are formally passed through and Indians cannot say that they did not receive anything.

Team dynamics

Due to no non-verbal or communication, it is difficult to know who is in your development team and how to communicate with each other. It is difficult to get to know the developers in India, as they give very short and formal answers. There is little or no transfer of knowledge.

Cultural differences

Everything needs to be explained at detailed level in order to let the Indian developers understand what is desired. If you have remarks, try to say this directly to the developer and not the whole group. Indian developers always say yes, even though they do not understand. Try to check whether Indian developers understand what they have to implement by asking questions about the requirements of the system.

Alter alignment attribute

Analysis design patterns can also be used at the quality attribute about patterns and not only design patterns.

Looking at Use-case Realizations, see what happens with an use-case after the requirements specification, and to check whether there are best practices to do this.

Case Study "Delta"

Report Interview Respondent A

Respondent	**A**	Interview date	**2-7-2007**
Role during the project	**Project Manager**		
Country where development took place	**India**		
Background/education	**Computer Science, Haagse Hogeschool**		
Number of development projects involved	**10+**		
Other important comments			

Syntax Dimension

Minimum requirement artifacts needed for design

Sometimes more details were needed to completely understand the customers' process. There was lack of understanding from the customer, as they were unfamiliar with the RUP methodology. Artifacts needed are Use Case Model, Glossary, Vision, SAD, Supplemental specifications, Use Case Realizations, & All Technical artifacts in RUP. No artifacts were missing; however, there was need for more detailed uses cases. There was too much open for own interpretation. Additional contextual information is needed in certain use cases in order to understand exactly what is desired.

Level of design needed in development location

During this project there was a strategy to do all technical design in The Netherlands. The reason for this was that we wanted to be sure that the system to be developed was managed properly and there were no dangers of making a wrong design. We knew that this restricts creative design.

In India they can also do a detailed technical design when they have gained enough knowledge about the whole project, the functional design and first technical designs. A good solution that works is to bring a few developers and the lead-developer to The Netherlands to exchange knowledge and information. The exchange of information and knowledge should be done immediately during the inception phase and not later, as this will prevent mistakes made by Indian developers.

Reuse of building blocks

Public-Key Infrastructure (PKI) of security company was used for security. The knowledge of the security components was exchanged by letting people from the security company show the component to LogicaCMG people directly. No understanding problems as only the Dutch developers used the PKI component and there was no need to transfer this knowledge to Indians. Next to personal help, specific training sessions may help to understand the components.

Consistent information

There were some minor inconsistencies, but this did not cause significant problems. During reviews of the artifacts most inconsistencies were dealt with. Specific inconsistencies were found during design and solved immediately.

Analysis and Design Patterns

A lot of design patterns were used, like Model View Controller that gave developers a clear structure to implement an already proven solution.

Gatekeeper

There was not really a single point of contact during this project. The architect did send the technical documents to India. The project manager in India communicated with project manager in The Netherlands. There was one Indian Lead Designer that came to The Netherlands to get to know how things are done here and to know the context of the project. Information about financial topics was open for many interpretations and it is difficult to communicate what information about the customer. Knowledge about financial processes of the customer is difficult to communicate.

Semantic Dimension

Reduce ambiguity

In use cases there was lack of information and knowledge in the financial domain. Certain financial categories were incorrectly used. Have a person in your development team who has sufficient domain-level knowledge of the customer.

Defining quality requirements

The number of hours of each function point was calculated incorrectly. Java projects need more time compared to Microsoft .Net projects. Security described in supplementary specifications was not high enough and was difficult to estimate.

Enough context of the surrounding

Most external couplings were specified in Interface documents. The software architecture document (SAD) also described enough context of the environment. The documents relating to environment context, like interface documents and SAD, need to be more detailed, as de developers in India cannot ask the customer or analyst directly in The Netherlands for additional information.

Pragmatic Dimension

Verification of requirements

During this project the requirements were not maintained in a tool, like Optimal Trace or RequisitePro. For us it was difficult to see whether the requirements were reflected in the design. It is possible to make use of use cases to and see if all uses were reflected in design. However, this is too time-consuming to do for this project. Using verification really needs proper tooling to quickly see whether the requirements are reflected in the design.

Required skills and experience in development team

There was no sufficient experience with RUP at the customers' site, which resulted in little understanding of the whole development process. Analysts have to speak the customers' language in order to let them understand the development phases and documents. Also people with specific domain knowledge of the psychiatry specialists were desired. During the project there were two people with proper knowledge of the psychiatry domain. What would really help is that one person at the customer attends a RUP essentials training, which helps to have a better understanding in future projects. However, experience shows that there is lack of interest from the customer to learn about RUP.

Changing requirements

After a preliminary version of the system there was a check if the system fulfilled the need of the customer. The customer indicated that the GUI was not as desired and needed to be changed. In addition, the level of security changed (PKI interfaces with smartcards). There was and additional requirement relating to the possibility that another psychiatrist could access the patient records when the helping psychiatrist is not available due to holidays for example. Additional functions were specified after there was more knowledge of the psychiatry domain. So the domain knowledge resulted in additional and changing requirements of the system. After a changed requirement, an analysis of the impact was made, the design was altered and the changes were planned to be implemented. Developers that were busy within the particular area of the system that contained the changing requirement ultimately implemented the new functions. Communication occurred using the following steps:

- Notice about the change
- Send altered design
- Discuss the technical design with India by means of teleconferencing

Availability team members

There was a time difference of 3,5 hours and we had to deal with this by means of proper planning. Furthermore, there are a lot of holidays in India. During this project there was a small problem during the delivery of the system, as not everybody was available due to a holiday.

The best way to deal with this is to take all holidays into account in the project plan in order to prevent that developers are unavailable when the system is going to be delivered.

Team dynamics

It is important that Indians come to The Netherlands to get to know each other in the development team. This decreases the tension between the people in your team and helps cooperation between them. They have a better understanding of each other and can communicate more effectively, as they understand what the other person is saying. Still this approach does not take away all barriers, but it surely helps to have a more dynamic team. Their bad English was also a restriction, as it was not always clear what they meant and whether they understood the use cases.

Cultural differences

A bottleneck sometimes was the language, as it was sometimes unclear whether the Indian developers really understood what you were saying. When they say yes that they understand something, you still do not know if they really mean it. Indian developers have a very social life, which makes them work less efficient compared to Dutch developers. Monitoring time and their productivity showed that Dutch developers work more efficiently. However, due to the strong social interaction, Indians might have a stronger Team spirit. There is high risk of job switching in India and this means that specific experience gained by Indian people might be lost on the long run.

APPENDIX F: OVERVIEW CULTURAL DIMENSIONS

Cultural Dimension / **Repondent**	Power distance	Individualism	Masculinity	Uncertainty avoidance	Long term orientation
Value in Netherlands*	**+/-**	**+**	**-**	**+/-**	**+/-**
Value in India*	**+**	**+/-**	**+**	**-**	**+/-**
1 (Dutch Team Case I)	+/-	+	-	+/-	+/-
2 (Dutch Team Case I)	-	+	+	+	-
3 (Dutch Team Case I)	NA	NA	NA	NA	NA
1 (Dutch Team Case II)	-	+/-	-	+/-	+/-
2 (Dutch Team Case II)	-	-	+/-	-	+/-
3 (Dutch Team Case II)	-	+/-	-	+/-	+/-
4 (Dutch Team Case II)	NA	NA	NA	NA	NA
1 (Dutch Team Case IV)	-	-	+/-	+	+/-
1 (Dutch Team Case III)	-	+	-	-	+/-
2 (Dutch Team Case III)	+/-	+		-	+
Overall Dutch Team	**-**	**+**	**-**	**+/-**	**+/-**
1 (Indian Team Case I)	+	-	+/-	+	-
2 (Indian Team Case I)	+	-	+/-	+	-
3 (Indian Team Case I)	NA	NA	NA	NA	NA
1 (Indian Team Case II)	+	+/-	+	+	+/-
2 (Indian Team Case II)	+	-	+	+	+/-
3 (Indian Team Case II)	+	+/-	+	+	+/-
4 (Indian Team Case II)	NA	NA	NA	NA	NA
1 (Indian Team Case III)	+	+/-	+/-	+	-
2 (Indian Team Case III)	+	-	+/-	+	-
1 (Indian Team Case IV)	+	+	+	+/-	+/-
Overall Indian Team	**+**	**+/-**	**+/-**	**+**	**+/-**

Legend:

+:	high
+/-:	medium
–:	low
NA:	not available

*** according to (Hofstede, G. last accessed June 21st 2007)**

APPENDIX G: IMPORTANCE OF ATTRIBUTES

Alignment Attributes \ Respondent	A	B	C	D	E	F	G	H	I	J
Required skills and experience in development team	3	4	4	4	4	2	4	3	2	3
Consistent information	2	3	4	4	4	3	4	4	2	4
Verification of requirements	3	4	3	3	3	4	4	3	3	3
Reduce Ambiguity	3	3	3	3	4	3	4	4	3	4
Minimum requirements documents needed for design	2	4	3	4	3	2	4	4	3	4
Gatekeeper	3	3	4	4	4	2	3	3	2	3
Cultural differences	3	3	3	4	4	1	3	4	3	1
Changing requirements	2	4	3	3	4	1	4	3	3	4
Level of design needed in development location	3	4	3	2	2	2	4	4	3	3
Reuse of building blocks	2	3	3	3	3	4	2	3	3	2
Defining quality requirements	2	3	4	3	3	2	3	3	2	4
Enough context of the surroundings	2	3	3	2	3	4	3	3	1	2
Team dynamics	3	3	3	2	2	1	3	3	3	3
Analysis and design patterns	2	2	3	2	3	1	2	3	2	1
Availability team members	2	3	2	1	2	1	3	3	2	2

Importance (Scale: 1 - 4)

1
2
3
4

Respondent	**Role**
A, G & H	Analyst
B,C & D	Designer
E, F, I & J	Architect

Thesis

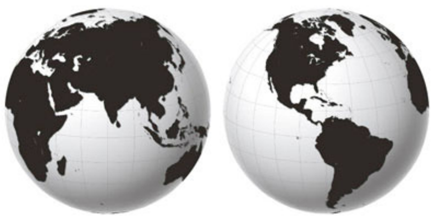
Alignment of Requirements & Architectural Design In a Blended Delivery Model

Reinier Kernkamp
Trainee Working Tomorrow

LogicaCMG – **Releasing your potential**

T:	+31 (0) 26 3765321
M:	+31 (0) 6 54305986
E:	0Hreinier.kernkamp@logicacmg.com

www.logicacmg.com



We've helped to generate business for utility companies worldwide.

We help support the missions of one third of the world's satellites.

We've helped forty of the world's top fifty banks to make more money.