Customer-Developer Interface

Exploring Best Practices of Communication with Ad-hoc Customers in Software Projects

Bachelor Thesis by

Jonas Wisbrant

June, 2001

LUND INSTITUTE OF TECHNOLOGY
Lund University

Q-Labs

Supervisor:
Associate Professor Per Runeson,
Department of Telecommunication,
Lund University

Advisor:
Peder Feldt, Q-Labs

Abstract

Problems in contract-based software development are often linked to problems in the relation with the customer. The objective of the study is to explore and model the organisational interface between developers and ad-hoc customers in software development projects. The objective is also to summarise, from the developer's viewpoint, different best practices for communication with customers.

The study uses a qualitative methodology. The customer-developer interface according to six established best practices for software development and six real world projects is assessed, described, summarised and analysed in terms of three theoretical perspectives:

- Process Model Perspective
- The Actors and Stakeholders Perspective
- Message Perspective

Objectives for a proper working customer-developer interface and *practices* for achieving these objectives are extracted from the established best practices. The findings in the best practices are compared to the findings in the real world projects. The usability of the three theoretical perspectives is successively evaluated in order to verify the results of the study.

- The three theoretical perspectives are shown being useful when assessing, describing and analysing the customer-developer interface of the best practises and the real world projects.
- A set of partly overlapping objectives for a proper working customer-developer interface is identified in the best practices.
- Practices for addressing the objectives are also identified. Interface objectives that have not been addressed by the best practices are identified in the real world projects.

Keywords: Acquisition, Actor, CMM, CRM, DSDM, Communication, Customer Relation, Gaps-model, Message, Process model, Procurement, Requirements, RUP, SA-CMM, Software, Stakeholder, Team Risk Management, XP

Acknowledgements

The work for this bachelor thesis has been performed at Q-Labs AB at IDEON Research Park in Lund, Sweden in collaboration with the Department of Communication Systems at Lund Institute of Technology.

To work in the creative atmosphere of Q-Labs, LTH Telecom and IDEON has been a privilege and a pleasure. I would like to thank my supervisors Per Runeson and Peder Feldt for their ability and willingness to give inspiration, answers and support throughout the work.

I would also like to thank my thesis colleague and roommate Pernilla Persson, partly for sending me many good email jokes and partly for sparring me in both English, logical thinking and table hockey.

Also a thank to all you Q-Labbers for being just as nice, competent and talkative as you claim to be.

Jonas Wisbrant

Lund, Sweden, 30 May 2001

Table of contents

Table of Figures

Why be surprised by something that can be clearly foreseen?

1 Introduction

1.1 Background

Software engineering is the activity of developing software within schedule, to a planned cost with expected quality. A mature development organisation is able to do this in a controlled way with limited deviations from project to project and without significant impact from surprises from within the organisation or from the project environment. But in the same moment an organisation decides not to develop a software product on its own, control of the development process is lost. The organisation can no longer use its own defined development process and its quality assurance program to guide the development. Or when an organisation decides to develop software on contract for another organisation, this control is also lost.

It is indeed very common for companies, government authorities and other organisations to acquire more or less complex software applications from different software developers. It is also very common for software developers to aim for an external end-user for its software. Each time one of these two situations happens, an interface between the two organisations is established.

Far too often, the delivered application will not reach the customer’s expectations regarding cost, functionality, quality or time of delivery. The reasons for these negative deviations from the expectations could be found almost anywhere in the software development process. As stated above, the basic objective for software engineering is to optimise software development according to cost, functionality, quality and time of delivery. Examining the entire development process for problems is too great a task, but an are with frequent problems is the communication between the customer and the developer during the project. The successes and shortcomings of this communication will have a significant impact on the development project. The subject for this study is the customer-developer interface.

This kind of study is often introduced with discouraging examples from the real world. The five billion SEK failure of the Ariane V rocket [Ariane V96], the luggage transport system at Denver International Airport [Gibbs 1994] and the overdosing radiation machine for cancer treatment [Leveson 1995, Appendix A] are famous examples of unsuccessful projects where the failures cannot be related to the customer-developer interface. The failure of the computer aided dispatch system acquired for London Ambulance Service in the mid-nineties [LAS 1995] can partly be blamed on deficiencies in the communication between developer and customer, by for instance missing the end-user’s needs and requirements.

1.1.1 The PPM Story

A rather new and very Swedish example to put on the list of discouraging cases is the failure of PPM[1] to acquire a web-based system for the administration of 5.5 million Swedes premium pensions. The story is not scientifically verified. It is based on an authority report [PPM 99-63] and some magazine articles [CS 1999:10], [CS 2000:38], [CS 1999:49].

The acquisition process of the system was initiated by RFV[2] in parallel with the establishment of the new authority PPM. The company that got the development contract, a well-known global corporation with 60 000 consultants, started with an object-oriented and very modern concept. When PPM was established it took over the acquisition responsibility from RFV. When the first prototypes where shown to the future system owners at PPM it turned out to be a system with wrong services and very little capacity for hundreds of thousands simultaneous users. The primary consequence of the misconceptions was that the premium pension reform, which had an immediate impact on the future pensions of half of the Swedish population, was postponed one year, involving changes of laws by the parliament and government. A crisis group with representatives from both the developer and RFV was established. The analysis showed that the chosen system architecture was not sufficient for the actual needs. In the meantime technicians at PPM had started to develop prototypes on their own, partly in order to clarify the system requirements. When there was only a half-year left to deployment, PPM decided to break the deal with the external developer and put efforts into the in-house developed alternative.

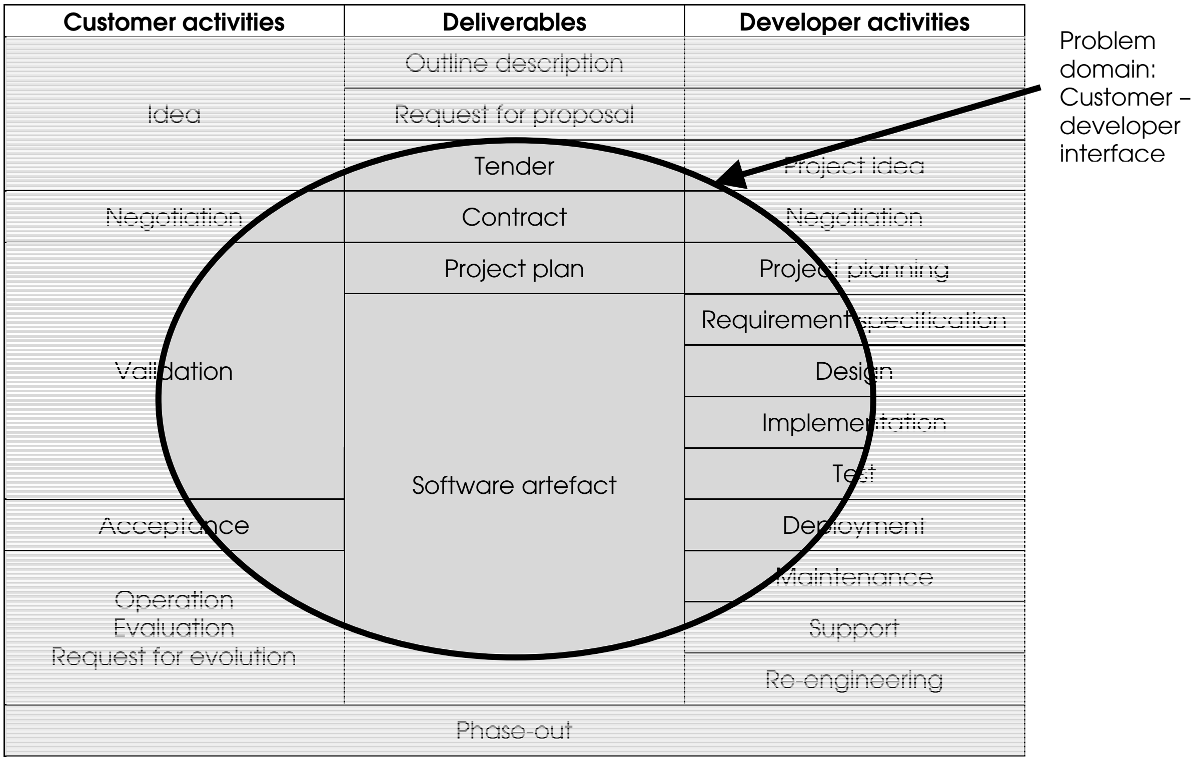
The lesson learned is that even though both parties were very experienced in their respective roles in the acquisition process; they had deficiencies in their communication and the validation of the requirements. The fault is mainly on the acquiring side. RFV requested a proposal when the system-owner existed only as a pile of paper. When PPM had learned to walk and talk it was too late to include its opinions. Sadly, these kinds of mistakes are likely to occur in such a unique situation. More surprising is the fact that the 60 000 strong contractor did not predict the problems inherent in the acquirer’s organisation. The contractor did nothing to prevent the problem. The contractor did not back out of the bid.

[1] Premiepensionsmyndigheten: The Swedish Premium Pension Authority

[2] Riksförsäkringsverket: The National Social Insurance Board

1.2 Objective

Figure 1 below illustrates activities and deliverables that in some form always appear in a software development project lifecycle. The circle in the middle describes the problem domain of this study, *the customer-developer interface*.



Customer activities	Deliverables	Developer activities
Idea	Outline description	
	Request for proposal	
	Tender	Project idea
Negotiation	Contract	Negotiation
Validation	Project plan	Project planning
	Software artefact	Requirement specification
		Design
		Implementation
		Test
Acceptance		Deployment
Operation Evaluation Request for evolution		Maintenance
		Support
		Re-engineering
Phase-out		

Figure 1: The problem domain – the customer developer interface.

One objective of the study is to explore a sub-class (see Section 1.4.1) of the interface between a customer and a developer in software projects. Another objective is to model and summarise how different best practices of software development handle the customer-developer interface.

The prime users of the result of the study are account managers or project managers in software development companies with a market in service oriented organisations. The developer's customer may for instance be someone that provides service or products by the Internet or through mobile phones. The result of the study would ideally be used when the account manager writes a tender for a software development project or when the project manager plans the project. The report from the study can inspire these persons to introduce or refresh the following questions into their agenda:

- What risks are we facing if we sign a contract with our prospective customer?
- How well suited is the customer to specify expectations, needs and requirements?

- Is it likely that the expectations and requirements will change during the project?
- Is the customer capable of validating the system?

The study will not answer these questions, but the report can provide give a framework for handling questions like:

- How should we best design our interface to the customer to mitigate fears regarding the customer's acquisition capability?
- Should we adjust our inner organisation or development process?
- Should we back out of the project?

Questions like these are traditionally treated in the risk management section of project management. The ambition here is to reduce the need for *subtle intuition* in favour of a more *structured approach* when identifying the risks.

Figure 2 below is a re-worked variant of Figure 1, page 13. The highlighted developer activities illustrate the viewpoint of the study and thought of end users of the report. More on this subject in Section 1.4.5 Scientific Perspectives on the Interface.

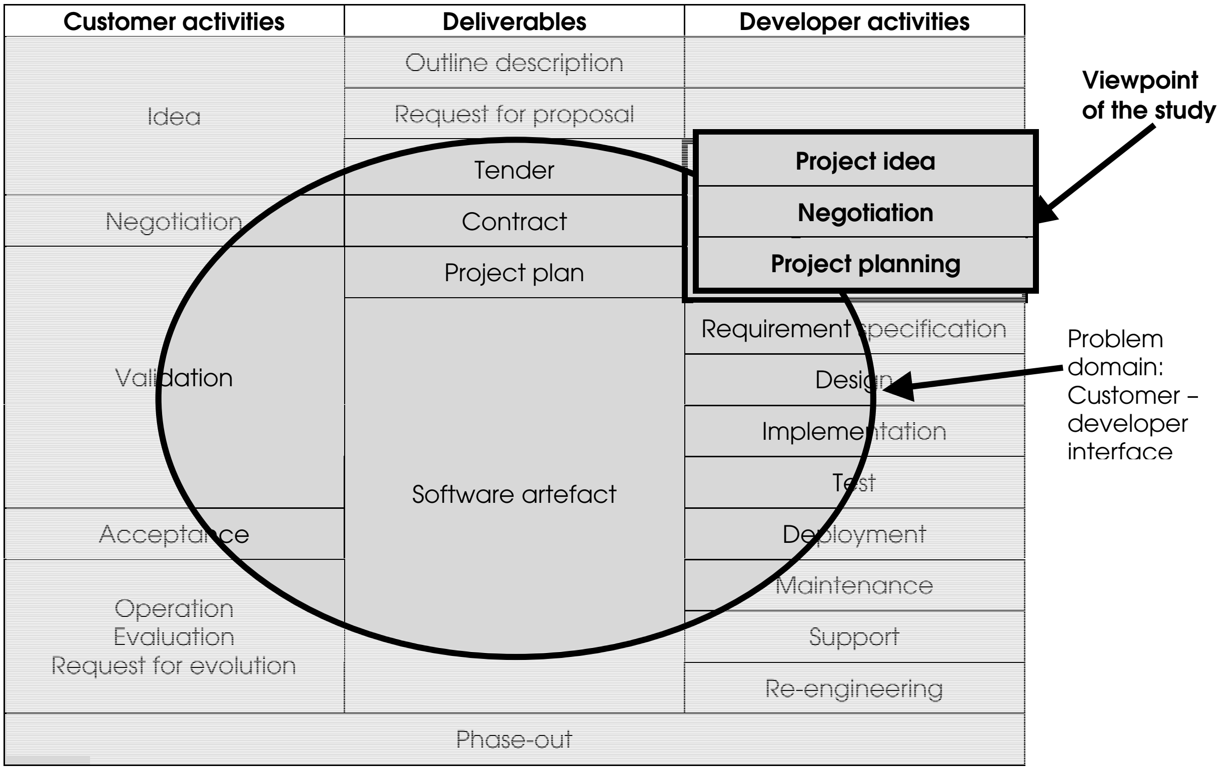


Figure 2: The perspective of the problem domain the customer-developer interfaces.

1.2.1 On the Control of Software Processes

A tradition within software engineering, e. g. in the software improvement models developed by Humphrey [Humphrey 1995] and SEI Capability Maturity Model [Paulk 1997], is built on the axiom “You can’t control what you can’t measure” [DeMarco 1982, p 3]. This tradition has developed a very complex and fertile methodology, which mainly involves statistical analysis of measurements of vital aspects of the software development process. This analysis makes it possible to verify whether an improvement of a development process reaches its objectives. It would be very pleasant to be able to contribute to that tradition by suggesting statistical measurable aspects of the customer-developer interface in order to strengthen the framework for process improvements. Unfortunately such an objective is very hard to fully accomplish, since the customer-developer interface is composed of two organisations of which the other one is hard to control. The result of a process improvement is hard to statistically verify if the parties have only a short time to build a relationship. Different corporate cultures may also make it hard or costly to share an improvement process.

1.3 The Problem Domain Terminology

There are a lot of almost synonymous terms flourishing in the literature of the problem domain. Before any definitions are made some of these termes are listed below:

- Acquisition – procurement
- Customer – acquirer – end-user – buyer – consumer – purchaser
- Software – programme –application – product – artefact
- Deliverer – supplier – contractor – developer
- Offer – tender – bid
- Contract – deal – agreement – settlement – arrangement
- Specification – requirement – feature description – outline description

Mostly it does not matter which of the terms is used, either because they are completely synonymous or because the context makes the meaning of the term obvious. This section discusses and defines three of the most essential terms of the study: the *customer*, the *developer* and *software*.

1.3.1 Customer

The term *customer* is chosen because of the chosen developer perspective for the study. From a developer perspective the term customer gives a proper attitude. The Software Engineering Institute (SEI) at Carnegie Mellon University has defined a *customer* according to what the customer does in a software acquisition project:

> “The organisation acquiring systems. The customer is responsible for:
> - defining requirements
> - obtaining funding
> - selecting supplier/contractor

- negotiating the contract
- accepting the product” [CMU/SEI-94-SR-5, p 4]

1.3.2 Developer

For this actor the choice of term stands between *deliverer*, *supplier*, *contractor* and *developer*. Higuera et. al. defines in a text about *Team Risk Management* the *supplier* as the

> “…organisation developing and producing the system and is responsible for implementing the requirements under the terms of the contract, which include cost and schedule.” [CMU/SEI-94-SR-5, p 5].

This is a straightforward definition of what a software supplier does in relation to the customer. However, in this study the term *developer* is chosen since it is the term that best applies to what the organisation mostly do and that is to develop software.

1.3.3 Software

IEEE Recommended Practice for software acquisition deals with three different categories of software products according to the degree to which the acquirer may specify the requirements [IEEE std 1062-1993, pp 1]:

1. COTS – commercial off the shelf software:
   This type of software is stable and well defined in terms of capabilities and documentation. There is a market driven need for the product and the developer is not willing to modify the product for a single customer.
2. MOTS – modified off the shelf software:
   MOTS are almost the same as COTS but some major tailoring towards customer-specific needs are to be made.
3. Fully developed software:
   This software is one of a kind. It has potentials for further development and the documentation is project specific.

If nothing else is stated, the term software in this study will cover the meaning of *MOTS* and *fully developed software*.

1.4 Limitations

The object of the study, *the customer-developer interface,* at a first glance seems to be possible to overview. It is not. It is necessary to make some limitations. This section deals with some major limitations and their consequences for the result of the study. Some other limitations, that might affect how to understand the results of the study, are dealt with on the fly. A second purpose of the section is to give a broader picture of the problem domain.

1.4.1 The Maturity of the Actors

An organisation that decides to let another organisation develop a software application creates an organisational interface between the two organisations. If both parties are well-organised and mature organisations, with a great experience in their respective role in the software acquisition process, it is quite possible that they have processes and procedures to handle the situation. The interface during the project has a good potential of success. But if one of the organisations is not experienced, there are a lot of pitfalls to be caught in. One can roughly divide the parties in a software acquisition into two categories; *mature* and *not mature*. Then there are four possible combinations. Examples of these combinations are given in the matrix of Figure 3 below.

		Software developer	
		Mature	**Not mature**
Software Acquirer	**Mature**	Established software consultants develops sub-components for a big telecom industry	Rapid growing internet Consultant company develops a multi-sender municipal web portal
	Not Mature	Established software development company develops a business management system for an ad-hoc customer	Small advertising agency develops a website for a local street kitchen.

Figure 3: Customer and developer maturity matrix

The prime target for this study are those developer-customer interfaces that are created in the highlighted lower left corner of the matrix in Figure 3 There are several reasons for this limitation.

Companies in the upper left mature-mature combination are well guarded by their own maturity. If, for instance, both parties are engaged in a CMMI- [CMMI 1.02d] or SPICE- program [SPICE] they already have instruments and procedures to guide them into a proper organisational interface or perhaps choose a more mature developer.

In the upper right part of the matrix, where the developer is not very experienced but the customer is, it is likely that the acquiring part is aware of the problems and thereby tries to control the interface.

In the lower right corner of the matrix, were none-of the parties has significant experience of the inter-organisational interface, there are risks for a great number of different problems, by which the interface aspect only are one. Furthermore has the developer no process context to fit an improved process into.

Left for this study is the lower left part of the matrix. An experienced software developer can be assumed to be aware of the problems arising when working with inexperienced or ad-hoc customers. Thereby the developing company might have an incentive to reduce the problem. At the same time these developers can be assumed to have some kind of development process framework into which potential improvements will fit. The fact that the customer more often than the developer is ad-hoc also puts a greater responsibility on the developer.

1.4.2 The customer and developer organisation

The communication between the customer and developer consists mostly of information about expectations, requirements, timing, responsibility etc. Facts and ideas are shuffled from one part to the other and back again. In many real life projects e.g. when developing for of-the-shelf-market or when the acquiring organisation is a sub-contractor for a bigger system, the end-user is not a part of the acquiring organisation. Another situation is when a developer, for some reason, might need to use a sub-contractor to develop a component in the software. If one, or both, of these situations occurs, there will be two or three interfaces to deal with in accordance with Figure 4 below.

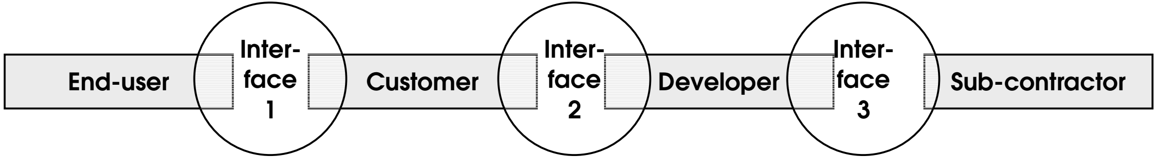


Figure 4: Multiple interface development

If it is problematic to communicate with a second organisation that one has a contractual relation to, it is obvious that it is an even more complex challenge to communicate with a third part through the second part. But even if the contents of the communication probably differs between the interfaces one, two and three of Figure 4, the increasing challenge mostly is a matter of level, not of nature. Therefore in the theoretical parts of this study it is assumed that the customer is in full control of the user-side of the development process and that the developing organisation is in full control of the developing side of the development process. This situation is described in Figure 5 below.

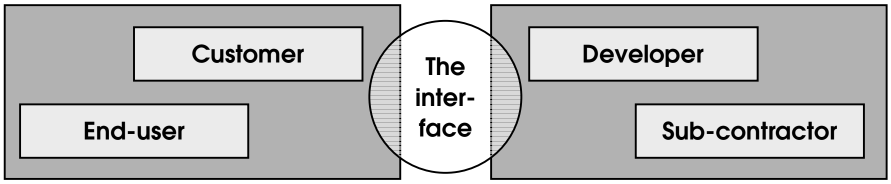


Figure 5: Single interface development

The inner organisations of the customer and the developer will be further discussed in Section 3.2 The Actors and Stakeholders.

1.4.3 Section of software lifecycle

Next limitation will be made according to the software lifecycle. Even if it is hard to prove, a software artefact may work forever. It may be used for decades or even longer. As a natural consequence of this it is possible for the customer-developer interface to be used during a very long period. This aspect should of course be thought of when planning a development project and thereby ideally also be included in this study. Unfortunately it would be very hard to trace software projects for decades. One could investigate project plans for how deeply they deal with customer-deliverer communication according maintenance or re-engineering ten years ahead, when yet unknown technologies, user-expectations and organisational trends rule the world. But to evaluate the project plans in relation to the project outcomes would be difficult. Therefore the theoretical parts of the study will exclude communication during the latter parts of the software lifecycle. A limit will be put somewhere at deployment/acceptance.

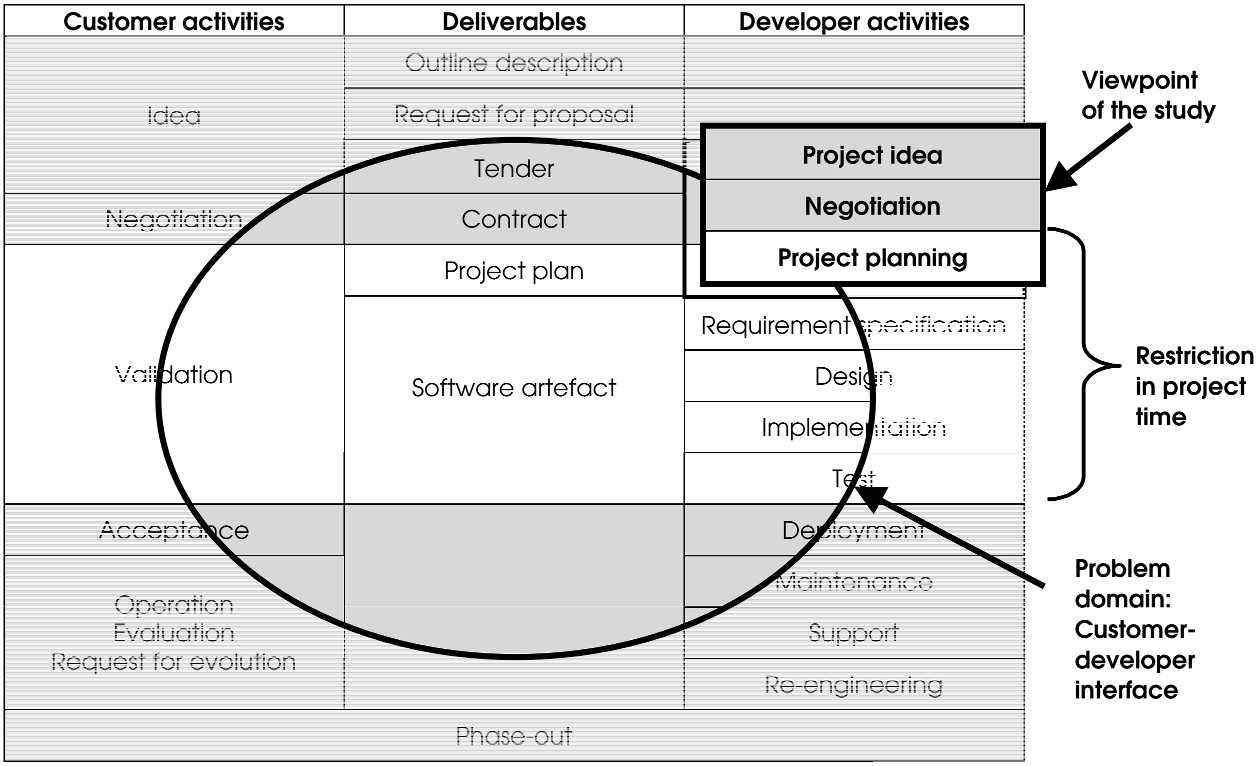


Figure 6: Generic view on a typical software lifecycle.

Figure 6 illustrates activities that in one form or another always appear in a software development project lifecycle. The study is restricted to the activities and deliverables localised in the white area in the middle.

1.4.4 Factors affecting the customer-developer interface

It is quite easy to come up with ideas for factors that in one way or another could stimulate, interfere or in some other way affect the functionality of the customer-developer interface. One is how the mutual relationship between the parties works. Other aspects are what kind of development processes that are to be used and what kind of product that is to be developed. Still other factors affecting the communication are the size of the project, the customer’s inner organisation and the corporate culture on both the developer’s and the customer’s side. Even if each of these aspects might have a very significant impact on the customer-developer interface the resulting model would probably become very complex. Such complexity would have two negative consequences. Firstly the model would both be hard to develop and to verify. Secondly the model would be hard to use. There are at least two ways for making a trade-off between *easy to use* and *easy to develop*.[3] A proper way of dealing with the problem is to choose those factors that can be regarded as positive in both cases.

Three of the factors or perspectives affecting the interface chosen in the study are:

- Process Model Perspective
- The Actors and Stakeholders Perspective
- Message Perspective

1.4.5 Scientific Perspectives on the Interface

It is quite easy to find many different realistic academic approaches or perspectives on what occurs in the interface between customer and developer. Examples are

- An **economist** would perhaps want to examine how much resource in form of time and money the two parties spend on their communication.
- A **psychologist** would be interested in the personal motives of the involved actors.
- The hands-on **software developer** would like to know how to determine when there will be an end to new requirements.
- A **lawyer** would start the study in the contract and investigate which of the parties that was to blame for a failure.
- An **ethnologist** would focus on differences in the corporate cultures or social context of the two parties and find a lot of problem there.

The list could have been made longer and moreover not so adequate perspectives of the problem domain e.g. chemical, ecological or astrological could have prolonged the list.

Since it is hard to apply all of the academic perspectives in the same study, a choice among them has to be made. A choice of perspective always means a choice of possible models and explanations. The choice can be made for several reasons. E. g:

1. The thought of end-user of the result. Who will gain from the result?
2. An idea of which hypotheses and explanations that are most relevant? Which perspective could be most fertile?
3. The competence and capability of the researcher and his/her advisors to work with the academic tools connected to the perspective. What questions is the researcher able to deal with?

The choice of a *project management perspective* (see Figure 2, page 14) are based on a mix of the first and the third reasons above, in combination with a belief that the perspective is fertile enough to motivate spending time and energy on the problem domain.

[3] This is by the way the same kind of trade-off that many software developers make - easy to code vs. easy to use.

1.5 Outline of the Report

1.5.1 Section 1: Introduction

This section (Section 1 Introduction) has introduced the reader to the background, objective, some terminology of the problem domain and some fundamental limitations for the study.

1.5.2 Section 2: Methodology

Based on some methodology literature two alternative approaches to achieve the objectives of the study will be described:

- **Quantitative** by statistical analysis of inquiry
- **Qualitative** by analysis of established best practices and case studies captureded by interviews with project managers

The qualitative approach is argued for, chosen and described.

1.5.3 Section 3: Theoretical Perspectives of the Customer-Developer Interface

The third section outlines three theoretical perspectives from which a customer-developer interface may be observed. The section also shows how the customer-developer interface may be described when observed from the theoretical perspectives. The perspectives are:

- Process Model Perspective
- The Actors and Stakeholders Perspective
- Message Perspective

1.5.4 Section 4: The Customer-Developer Interface According to Best Practices of Software Development

Section four gives a summary and discussion of how the customer-developer interface is treated by some established best practices. The chosen sources are:

- SEI Software Acquisition Capability Maturity Model – SA CMM
- SEI Team Risk Management – SEI TRM
- The Gaps-model
- Rational Unified Process – RUP
- Extreme Programming – XP
- Dynamic Systems Development Method – DSDM

Different *objectives* for a customer-developer interface are identified as well as *practices* for achieving these objectives.

1.5.5 Section 5: Customer-Developer Interfaces in Real Life Projects

In order to verify the findings in the established best practices, the fifth section of the report deals with how six real world customer-developer interfaces has been assessed and described (Appendix A – Cases Of Customer-Developer Interface) and analysed in the light of the three theoretical perspectives. Thereafter the findings in the real world cases are compared to the findings in best practices.

1.5.6 Section 6: Summary and Conclusions

The sixth section summarises the study and its findings. The objectives of the study are evaluated and some conclusions presented.

1.5.7 Section 7: Further Research

Section seven throws light upon some unanswered questions and presents ideas for how those questions may be tackled.

2 Methodology

The objective for the study as described in Section 1.2 is to *explore the customer-developer interface* and *model and summarise best practices* of customer developer communication in development projects. This is easier said than done. A definition of the problem domain is done (Section 1.3) and some major limitations to the area are described (Section 1.4). The question for this section is to describe how new and relevant knowledge is to be created, validated and presented in a consequent, consistent and verifiable form. In other words - what methods are to be used? The methodological thinking in the section is based on what is written in Holme et al "Forskningsmetodik – Om kvalitativa och kvantitativa metoder"[4] [Holme1997].

The two basic strategies for a scientific study are the quantitative and the qualitative approach respectively. To *find out* and *verify* good practices of how to interact with an ad-hoc customer in a software development project one could use the quantitative approach. One way could be to try to measure variables in the communication process that is likely to affect the successes and shortcomings of the development process in the large, or just the customer communication aspect in the small. With statistical computation it would then be possible to show the correlation between the variables (cause) and the outcome (effect). Moreover, it would also be possible to identify which of the variables that has the most significant impact and thereby would be most important to avoid or accentuate when planning the project.

In opposition to the quantitative approach, the qualitative approach does not aim for statistical verifiability. Here the objective is rather to find variation, structures and context in order to get a deeper understanding of the problem domain. The qualitative researcher is allowed to affect the studied object, which is not desired in the quantitative study.

This section describes one possible quantitative and one possible qualitative methodological strategy for meeting the objectives of the study as described in Section 1.2 and presents arguments for a choice of strategy.

2.1 Outline for a Quantitative Approach

Another way of describing the quantitative methodology is that the researcher raises some hypotheses, e g that some phenomenon in the customer-developer interface affects the outcome of the project in some certain direction. The next step would be to statically verify whether the hypothesis is true or not. Figure 7 below lists examples of such measurable hypotheses on the problem domain of this study combined with suggestions of measurable variables.

[4] Translation: Research methodology – on qualitative and quantitative methods

Hypotheses	Ideas for measurable variables
The fewer stakeholders at the customer side, e g classes of end-users, sponsors, system responsible etc that is affected by the software to be developed the better will the project outcome be.	• The number of stakeholders is measured by counting them. • The project outcome are measured by the end-user final system opinion questionaries.
The more messages between the customer and developer during the project that is exchanged outside what was defined in the project plan the worse will the project outcome be.	• The number of messages is measured by statistics on meetings, mail, phone calls and use of change request tools.
The more contact-points between the parties there are the better the project outcome will be.	• The number of contact points is measured by interviewing the participants on their contacts.

Figure 7: Hypotheses and variables for a quantitative analysis of the customer-developer interface

An approach like this would have the great effect that the study would come up with statistically verifiable truths. But there would also be some complications. In order to make the results statistically reliable, the study would for instance have to analyse a large number of projects. The environment would also have to be controlled. If a stipulated positive correlation between two variables were identified, it would also have to be shown that the second variable depended on the first – not in the opposite direction or both variables depending on a third variable. An outline for such a quantitative study could be:

1. Identify hypothetical causes and effects in the customer-developer interface.
2. Show and argue for their dependency relations.
3. Find assessment points in real life projects.
4. Argue for the connection between the assessment points and their respective *cause* or *effect*.
5. Assess metrics from real life development projects.
6. Compute the statistical correlation, significance.
7. Compute the probabilities that the hypothesises about causes and effects (point 1) are true or not.

2.2 Arguments for a Qualitative Approach

Holme et al [Holme1997] summarise the characteristics of the quantitative and the qualitative approach in Figure 8 below.

Quantitative methods	**Qualitative methods**
1. Precision: the researcher wants fair reflection of the quantitative variation.	1. Accuracy: the researcher want to maximise the coverage of the qualitative variation.
2. Limited information on many units. A broad investigation.	2. Much information about few units. A deep investigation.
3. Systematic and structured observations. E. g. questionnaires.	3. Unstructured interviews with open questions.
4. Interest in common features, average the representative.	4. Interest in the unique, differences, abnormal.
5. Distance to the object: data collection is parted from the studied object.	5. Closeness to the object: gathering of data is done close to the studied object.
6. Interest in separable variables.	6. Interest in coherence and structure.
7. Description and explanation.	7. Description and understanding.
8. Audience or manipulator: the researcher watches the phenomenon from outside and strives for the roll as observer. Variations in the object can be manipulated.	8. Participator or actor: the researcher observes the phenomenon from inside. He knows that he affects the result by being present. He may interact.
9. Me-it-relation between the scientist and the object for the study.	9. Me-you-relation between the scientist and the object for the study.

Figure 8: Characteristics for qualitative and quantitative methods (Holme1997 p 78)

Recall Section 1.2 Objective. In the perspective of the objective of the study, both methodological approaches are absolutely possible to use, but the qualitative approach is chosen. Primarily this choice is made for practical reasons. A serious quantitative study of the customer-developer interface could, for instance, be done in either of the following two ways:

- In one alternative, very much time and effort would have to be devoted to find a very small sub-set of the problem domain. An example of this could be *one project with a very long-term development contract with a specific customer*. In such case the study could first find and specify assessment points, then assess process variables and perhaps end up with a process improvement plan. This alternative would presumably be very interesting and valuable for the involved parties. But it would also face the risks of not finding the right project. It would be hard to follow over time and it would not allow extracting generic conclusions valid outside the involved organisations.

- In a second alternative a lot of people, organisations, time and money would have to be engaged in the study, for instance a mass survey about attitudes, experiences and methods in the customer-developer interface. Such study would, if properly carried through, give us valuable knowledge and give conclusions valid in a considerable part of the software society. Probably it would be a real hit at software engineering conferences. Unfortunately this second alternative in the first place would be hard both to finance and to finish in a limited period of time. Second, and most important from a scientific point of view, in order to formulate the hypotheses and casual connections that would be the foundation for the quantitative study, the study probably would anyhow have to be started up with a qualitative study like the one outlined in the next section. More on this in Section 7 Further Research at page 86.

2.3 Outline of a Qualitative Approach

A possible qualitative approach for *exploring the customer-developer interface* and *model and summarising best practices* of customer developer communication (see Section 1.2 Objective) will be given here. By studying real life project plans, process policies and project reports and by interviewing customers and developers about their experiences, it should be possible to extract knowledge and experiences of good and bad features of the customer-development interface. By comparing the experiences from different projects to each other, as well as to known and verified best practices of domains closely related to the customer-developer interface, it should be possible to picture relevant causes and effects. With this approach, the researcher is fully aware of the fact that each instance (a customer-developer interface in a software project) lives in a unique context out of control from the researcher.

To be able to describe the customer-developer interface in real life projects in a way that makes it possible to compare them, both to each other and to the related best practices, the studied interfaces have to be streamlined in some way. One way of doing the streamlining, is to describe the customer-developer interface in the light of a theoretical perspective, showing static and dynamic features that could be expected to appear. This perspective could then be used to describe both the real life projects as well as the related best practices in a form that makes it possible to compare them to each other.

The outlined approach, however, involves a risk. Suppose that the theoretical perspective fails to capture and model relevant aspects of the best practices and the real life projects. Then the summation and comparison of projects and best practices would be misguiding.

In order to reduce the risk of choosing the wrong perspective – three perspectives are chosen instead of one. From the objective and the viewpoint chosen in this study it is adequate to select perspectives close to the environment of the project manager or software engineer.

- Process Model Perspective
- The Actors and Stakeholders Perspective
- Message Perspective

The choice has two benefits related to the arguing in Section 1.4.5 at page 20. One is that the perspectives are fairly easy to use. The other is that answers are easy to understand for the thought of audience.

Before the study is made there is no way to verify that the three chosen perspectives are better than others are - or good enough. Therefor, the study has to be supplemented with a strategy for answering the question whether the perspectives cover the relevant parts of the problem domain or not. Otherwise it is impossible to know if the qualitative *exploring of the interface* and *summation of best practices* gives maximum description of the qualitative variation. See Figure 8 upper right field.

One way of answering the question is to, when exploring the best practices and real life projects, continuously look for features not covered in the models. Was something left out? This is hard to do, but it is also hard to come up with a better strategy.

On the other hand, the risk of missing relevant features opens for a slightly shift of the question of the initial objective. The mission was to *model* and *summarise best practices*. Now it turns out to be, not drawing a model of *best practices*, but drawing a model of *the customer-developer interface*.

The consequences of the reasoning above are that the exploration, the description and the comparison of best practices and real life interfaces are not enough. A minor hypothesis that has to be dealt with has emerged:

> If a customer-developer interface is described by its process model, its actors and stakeholders and its messages, this description is all-embracing enough to give a relevant understanding of how to plan and evaluate a customer-developer interface.

A way of verifying the hypothesis is to use the theoretical perspectives when *capturing, describing* and *comparing* the best practices and when *capturing, describing* and *analysing* the real life projects. If these activities are successful and if there are not too many missing features in the best practices and real life interfaces, the hypothesis can be said to be true. If the hypothesis turns out to be true then the exploring, describing and comparing parts of the study also is very relevant. Otherwise the models have to be rebuilt or complemented in some way.

This approach outlines a concrete study like the following:

1. Outline and describe theoretical perspectives for capturing and describing static and dynamic features of the customer-developer interface:
    - Process Model Perspective
    - The Actors and Stakeholders Perspective
    - Message Perspective
2. Study how the customer-developer interface is described by best practices (known and verified software development processes).
3. Describe the best practices as shown by the theoretical perspectives.

4. Ask if the theoretical perspectives capture the essence of the best practices? List aspects that were left out.
5. Extract objectives and practice that can be related to the customer-developer interface.
6. Assess some real life projects through analysis of project documentation and interviews.
7. Describe the projects as shown by the theoretical perspectives. List aspects that could not be captured by the theoretical perspectives.
8. Analyse the real life projects and compare them to each other and to the best practices. Verify if the three theoretical perspectives were useful for assessing real life projects.
9. Analyse the missed *objectives* and *practices*.
10. Evaluate the objective of the study
11. Draw conclusions around the question: Can the theoretical model together with the best practice guide us to a better understanding of how to plan the interface to ad-hoc customers?

Recall the objective once again. *Exploring the customer-developer interface* and *model* and *summarise best practices* of customer developer communication

The theoretical perspectives together with the *summarised best practices* will constitute a *model* for how to avoid problems in treating the ad-hoc customer. If the list of missed aspects turns out to be very long or very important, the models have to be rebuilt or completed. Suggestions of how this may be done, will be made in Section 7. The *exploration objective* of this study will anyhow be met. The gathered material could perhaps furthermore be used as an interesting input to or inspiration for a future quantitative analysis of causes and effects in the customer-developer interface. More on this will also be discussed in Section 7.

2.4 Summary

Based on some methodology literature two alternative approaches for achieving the objectives of the study has been described:

- **Quantitative** by statistical analysis of inquiry
- **Qualitative** by analysis of established best practices and case studies fetched by interviews with project managers

The qualitative approach was argued for, chosen and described.

3 Theoretical Perspectives of the Customer-Developer Interface

Abstracted to a very high level, software development is about *transferring* and *transforming* information, experiences, knowledge and visions about the phenomenon and the rules in the future environment of the software. Software engineering guides both the transferring activities and transformation activities. The *transferring* is done by communication. Hence in all software projects there is a need for communication. Most of the communication concerns *what* the developed software is expected to accomplish. Some of the communication deals with *how* the software is to be developed. Whenever the software development is split on two or more organisations there will automatically be a need for communication *between the organisations*. The communication passes through the customer-developer interface.

What kind of information that should be exchanged, when it should be exchanged, under which form and by whom, is closely related to the answers of questions like:

- What kind of development processes is to be used?
- What kind of product is to be developed?
- How is the relationship between acquirer and developer?
- How large is the project?
- How are acquirers and developers organised?
- Which are the main actors and their stakes?
- How do they communicate?

How a phenomenon is described is, among other things dependent of the viewpoint or perspective of the observer. To get a detailed and useful picture of a complex phenomenon it is often useful to observe the phenomenon from different perspectives. Each perspective gives a unique description. Descriptions from different perspectives complement each other to a more complete and complex picture of the phenomenon.

This section presents three different theoretical perspectives of the customer-developer interface. Each perspective is connected to and exemplified with a description of features and phenomenon that the use of the perspective can be assumed to bring about. The perspective and its related description can in combination be regarded as a tool for describing and analysing the customer-developer interface. The three perspectives are:

1. **Process model perspective**
   The process models uses macro perspective, and result in a description of the customer-developer interface based on entities like high level activities, high-level deliverables and the ordering of and the relation between these high level activities and deliverables.

2. **Actor and stakeholder perspective**
   The actor and stakeholder perspective examines the customer-developer interface from in between the micro and the macro level. This perspective is more static over time than the process model perspective. The use of this perspective at the customer-developer interface results in descriptions of categories of people, departments or organisations participating in the software development process, why they are there and what objectives they can be assumed to have.

3. **Message perspective**
   The *message perspective* studies with the customer-developer interface at a micro level. Just as in the case with the perspective of process models, it is also dynamic. The achieved picture consists of what information that is transferred, between whom, when and how?

Each of these perspectives forms frameworks that help capturing the customer-developer interface in a detailed way. Together they form a tool for describing and analysing the customer-developer interface. In later part of the study, they are used as a mainframe in order to assess, describe and analyse both established best practices and real life customer-developer interfaces.

Remark! The perspectives chosen are by no means bound to the situations where the software development is split on two or more parties. Neither does the maturity of the involved organisation matter. This mean that they probably also can be used to analyse processes where a software system is developed within a single organisation.

3.1 Process Model Perspective

This section describes the customer-developer interface in the perspective of the project’s development process model.

A software process model is

> “a simplified representation of a software process, presented from a specific perspective.” [Sommerville2001, p 8]

Sommerville describes four different categories of such development process models. All four models are built around the fundamental activities:

- Software specification: Specification of software functionality
- Software design and implementation: The development of the software
- Software validation: Confirmation of that the software meets customer expectations
- Software evolution: Continuos evolution to meet customer needs

The four models are:
- Waterfall process model

- Evolutionary process model
- Formal system development
- Re-use oriented development

The choice of process model has a significant impact on the customer-developer interface and before discussing that impact, the following subsection shortly describes the four models.

3.1.1 Waterfall Model

The waterfall model is illustrated in Figure 9 below. The process model partitions the development project into separable phases where one phase produces the input to the next phase and each phase verifies that it produced what was stipulated by the former phase.

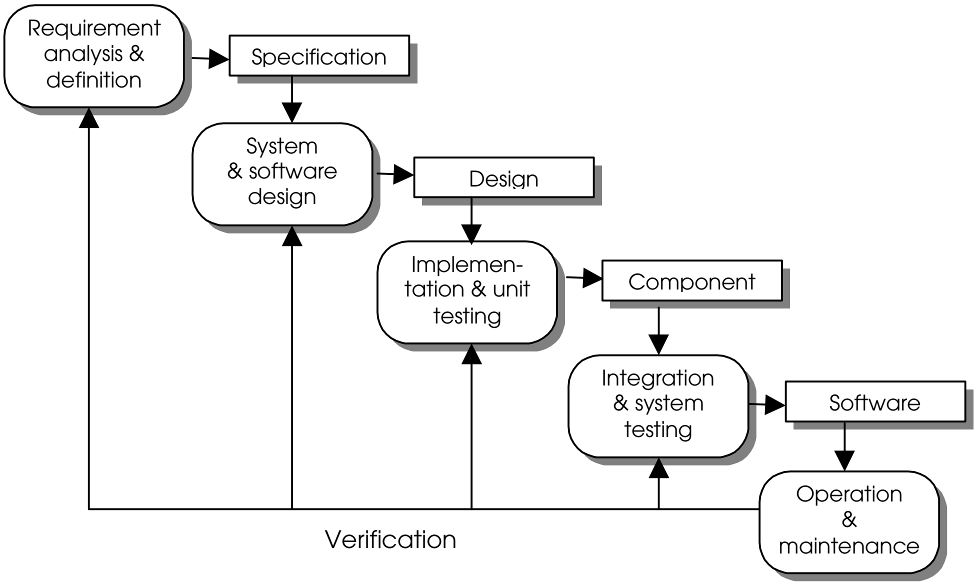


Figure 9: Waterfall development process model (Sommerville 2001, p 45)

3.1.2 Evolutionary Development

In the evolutionary development model the project starts with a very loosely defined outline of what is to be developed. Figure 10 below describes how an initial version of the software is, in co-operation with the customer and through several intermediate versions, successively develops to a final version of the software. Specification, development and validation activities are done in parallel on different parts of the system.

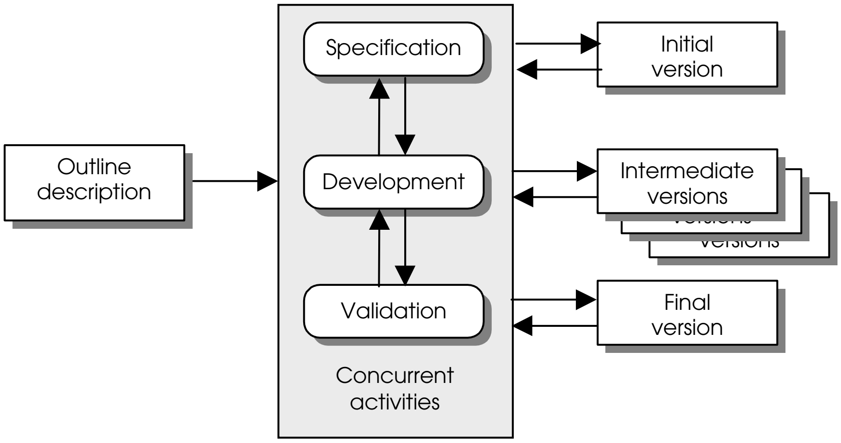


Figure 10: Evolutionary development process model (Sommerville 2001, p 47)

But Sommerville also identifies some different categorise of the evolutionary process models. The *exploratory process model* starts with mostly well known requirements. The reason for the evolutionary approach is explore possible technical solutions. A second variant is the termed *throwaway prototyping*. Here is the objective to by trial-and-error together with the customer, explore and find out the requirements of the system. The third variant is *incremental development*. The main difference between this and the two former evolutionary models are that the system on the start of the development are reasonable well specified. The first increment might for instance carry the most important features and be up running long before its successors. The incremental model has much in common with the waterfall model e g the need for stringent and baselined specifications and a controlled change management system.

3.1.3 Formal Systems Development

The formal system development model has much in common with the waterfall model. The big difference is a more stringent conversion from one phase to the next phase. The content of one specification is mathematically derived from its precursor. This model makes it very difficult to change requirements during development. But if the requirements are changed or added there should be a good possibility to check for negative consequences on other parts of the software product.

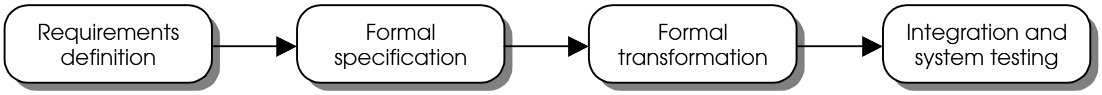


Figure 11: Formal systems development (Sommerville 2001, p 48)

According to how the process model affects the customer-developer interface, the formal development can be regarded as a sub-class to the waterfall model.

3.1.4 Reuse-oriented development

The fourth Sommerville process model is *reuse-oriented* and used when the customer or developer assumes the software can be constructed from existing components or COTS. A vital consequence of a reuse strategy is that there probably has to be some trade-offs between the desired features and what the market has to offer at a reasonable cost. This reuse oriented process model could be a sub-class to both the waterfall model and the evolutionary models. The difference lies mostly in the need to modify early requirement specifications.

3.1.5 The Waterfall-Evolutionary Axis

There are no exact mach between how things are done in real life and process models on a paper. But the interface between customers and developers looks absolutely different in the process models and their variants. Regarding the need for communication between the two parties it is easy to build a scale from the formal system development model in one end to the throwaway prototyping evolutionary model in the other end. A planned incremental development process can be placed somewhere in the middle.

Figure 12 below illustrates briefly this phenomenon. In the formal systems development the customer sends a very strict specification to the developer in the beginning of the project and receives a fully developed (derived) and tested software at the end. In the evolutionary development project the communication *between the parties is continuos* and perhaps more emphasising than within the respective organisations.

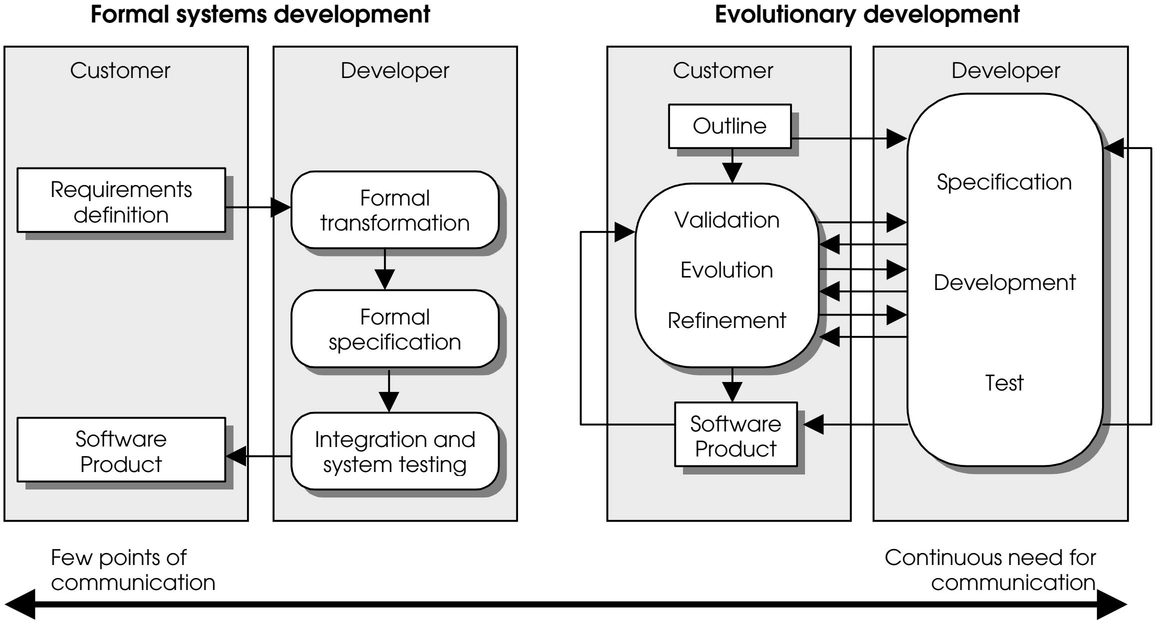


Figure 12: Process model extremes

There is no reason to believe that there is any difference in the amount of information about the software to be developed that needs to be transferred and transformed at the two extremes of the axis. But since each information occasion has to be accompanied with more or less overhead information (handshakes, scheduling, synchronisation etc) the total amount of information floating around can be assumed to be higher in the right part of the axis than in the left part. In other words, the number of communication events during the project increases continuously when moving further left along the axis. This means that a development project using an evolutionary process model has to be prepared to handle both a more intense and a more complex flow of information between the parties, than if the development process was organised according to the formal development model.

3.2 The Actors and Stakeholders Perspective

This section presents the customer-developer interface in form of actors and stakeholders. At an organisational level the basic actors of a software development project are the acquiring and the development organisations. In order to describe and analyse the actors and stakeholders of the customer-developer interface it is necessary to, on one hand look wider and include parts of their environment, and on the other hand, divide the basic actors to into smaller parts.

In major projects it happens that the acquiring organisation engage a supporting contractor to assist in matters like selection of developer, verification or system validation. Occasionally, the developer uses one or more sub-contractors. This happens for instance when the project runs short of scheduled time, special competence or other resources. [Marciniak 94, p 15].

The prime organisations – the customer and the developer – consist of departments, sections and individuals – each with unique or individual roles and special interest in relation to the software development project. In those cases the actor also can be said to be a stakeholder. In some cases the actor is a person. In some other cases the actor is a company department. Sometimes one actor represents several different – and in worst case conflicting interests. Figure 13 below describes, on *an organisational level*, important actors and stakeholders on the software market.

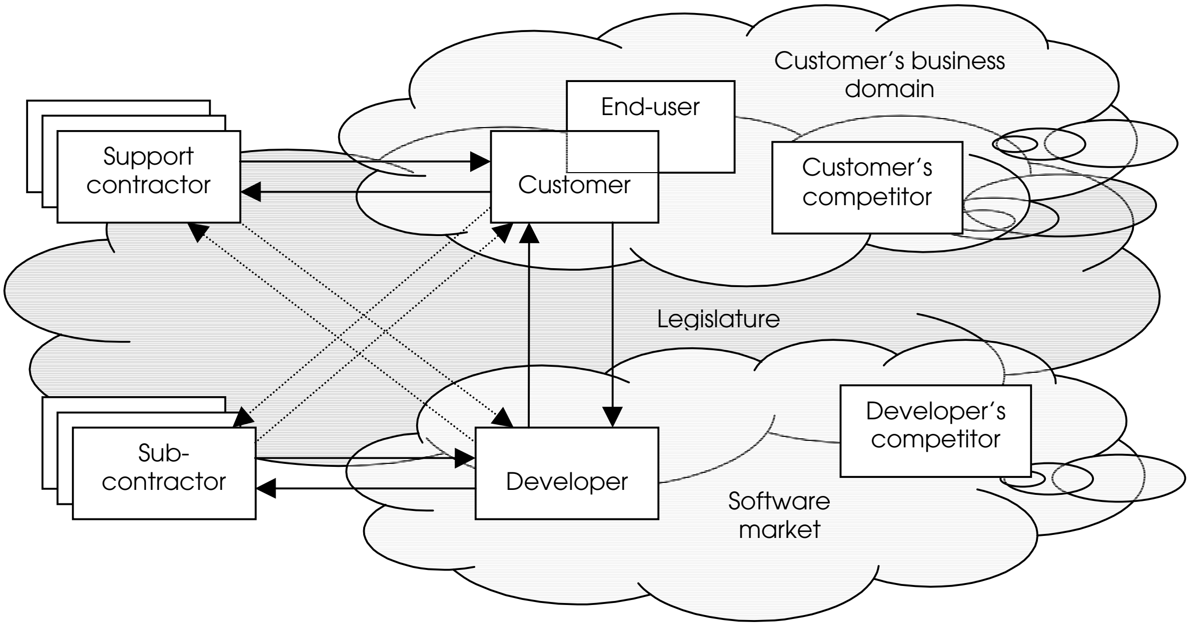


Figure 13: Actors and stakeholders on the software market

The actors in the environment have no direct impact on the customer-developer interface during the development. But they may have some impact before and after the project or indirect effect during the project, which are arguments for keeping some of them in mind:

- **Competing software development companies** will try to get the contract in competition with other software developers. Mostly this competition is accomplished with fair methods on factors as prize, quality and confidence.
- **The competitors of the customer** will try to gain market shares from *the customer*. One possible way of doing that is to acquire better business supporting software.
- **Legislature** has an impact on the acquisition process by setting the rules and giving a mainframe for how a contract should be formulated.
- **Official authorities** are often big acquirers of software and as such they have high demands from political institutions on the quality of their acquisition process and can thereby perhaps serve as a model for a good acquisition process.

In some management literature the description of the environment is much more penetrating, including cultural values, financial institutions, plagues, weather, lobby organisations and neighbours etc. In this study they are ignored since they are believed to have very limited impact, both direct and indirect, on the customer-developer interface.

The purpose of the rest of this section is to describe the head organisational and inner actors and their expected stakes in the software development process.

3.2.1 Actors in the Customer Organisation

As a whole, the customer organisation has the interest of doing a good and effective job on its business domain at a low cost. But if the software project is big there will probably be a lot of people involved in the acquisition process – each with a unique role, responsibility and interest.

Initiator
The initiator gives voice to the need of a new software application. The stakes might be a combination of personal ambition and a vision of how to make the business more successful.

The sponsor
The sponsor of the development project will provide capital for the development project. The sponsor will also take the economical risk based on information and estimations from other parts of the organisation.

The acquirer
The acquirer acts as representative for the acquiring organisation by formulating a bid for request, receiving bids from software developing companies, analysing those bids on behalf of the sponsor and/or system owner, and making decisions during the development process.

The system owner
The system owner will be responsible for the developed software once it has gone into use. The main stake is to get a well functioning system with low operating and maintenance costs.

IT-administrator
The IT-administrator deals with compatibility with parallel systems, in-house maintenance, local support and updating the systems for new users etc. During the development project the administrator will support with information about the systems future technical environment. The main stakes of the administrator is to avoid technical conflicts and get a smooth well operating system.

The end-user
The end-user actor has the real knowledge of and competence on application domain. If the software project is large and the customer organisation is big it might be possible to divide the end-user into different sub-categories. Expert end-user will be engaged in the work with requirement elicitation and validation activities. Normal end-users will use the final system and for instance act as guinea pigs by trying out and give feed-back on prototypes for a user interface.

At one extreme one person could represent all of the actors and their stakes at the customers site. For instance a freelancing consultant calling her hacking neighbour to develop a minimal database for her client records. On the other extreme 15 full time persons, whose only role were to gather feedback on a prototype of a large-scale business system, could for instance, represent the end-user.

3.2.2 Actors in the Development Organisation

The developing company has, in a way, the same overall goals as the acquiring company, *to do a good and effective job on its business domain at a low cost*. The difference is that business domain now is specified to software development. Thereby is it possible to further specify the stakes of the developer. The main stakes of the developing organisation is to *win the software-development contract* and fulfil it according to *price*, *schedule* and *quality* with an economic *surplus*. But, just as in the customer organisation, it is possible to identify several actors with unique roles, responsibilities and stakes. These are described in the following sections.

The project manager

The project manager is responsible towards the customer for delivering the developed software in time, with a specified quality. The responsibility towards the development company is to meet the customer's expectations within given budget. This is done by influencing the project plan, staffing and risk management etc. and by leading the project.

The creator of tender

The creator of tender is the actor who, based on information from the customer about what, when and how etc., formulates a tender, negotiates with the customer and finally signs a contract on behalf of the developer. The creator of the tender has two conflicting objectives. One is to win the contract. The second objective is to make that contract *a good contract*, in the meaning high revenues and low costs. In some organisations the creator of the tender is the same person as the project manager. Then a trade-off solves the conflicts. In a case where the roles are played by two different persons, there will be a high risk for surprises waiting for the project manager, if the creator of the tender underestimated the need for resources for the development.

Development

The development staff can be divided into several different sub-categories. Their common stake is the same as the project managers' - to deliver a software product with expected quality, on time and within budget. But each category also has some mutually different interest or needs – both generic and according to the customer interface.

- The **requirements engineer** needs very good access to the end-users both in order to elcitate detailed requirements and to validate the requirements specification to ensure that it is equivalent with actual needs of the end-users. In case of disagreement among the end-users the acquirer is called upon in order to conciliate.
- The **designer** needs both the end-users and the acquirer to validate the software design and overall structure. The designer sometimes also needs to directly

communicate with the acquirers’ IT –administrator to discuss conflicts or technical interfaces with existing systems.

- The **programmer** has seldom any need for direct communication with the customer organisation. If the software design seems odd, the problem is best broken with the designer who in turn might contact the customer if needed.
- The **tester** is in a broad sense responsible for ensuring that the requirements specification is fulfilled. If there appears conflicts among the requirements during a system test, there will be need for consulting the customer organisation. To make an acceptance test the customer organisation has to meet up with at least the acquirer, system owner and some representative non-expert end-users.

Product support
Product support has a broad scope of developing user manuals, train the end-users and give telephone support. This means a lot of contacts towards many different customer actors.

Quality manager
The quality manager monitors, evaluates and calibrates the software development process in order to ensure that the quality of the software reaches its objectives.

3.2.3 The Actors and their Interconnections

All the actors, stakeholders and roles in the two previous sections will communicate regarding the software project with other actors in or outside the project. Figure 14 pictures the actors and what might be the most frequent or relevant communication lines in different phases of the project.

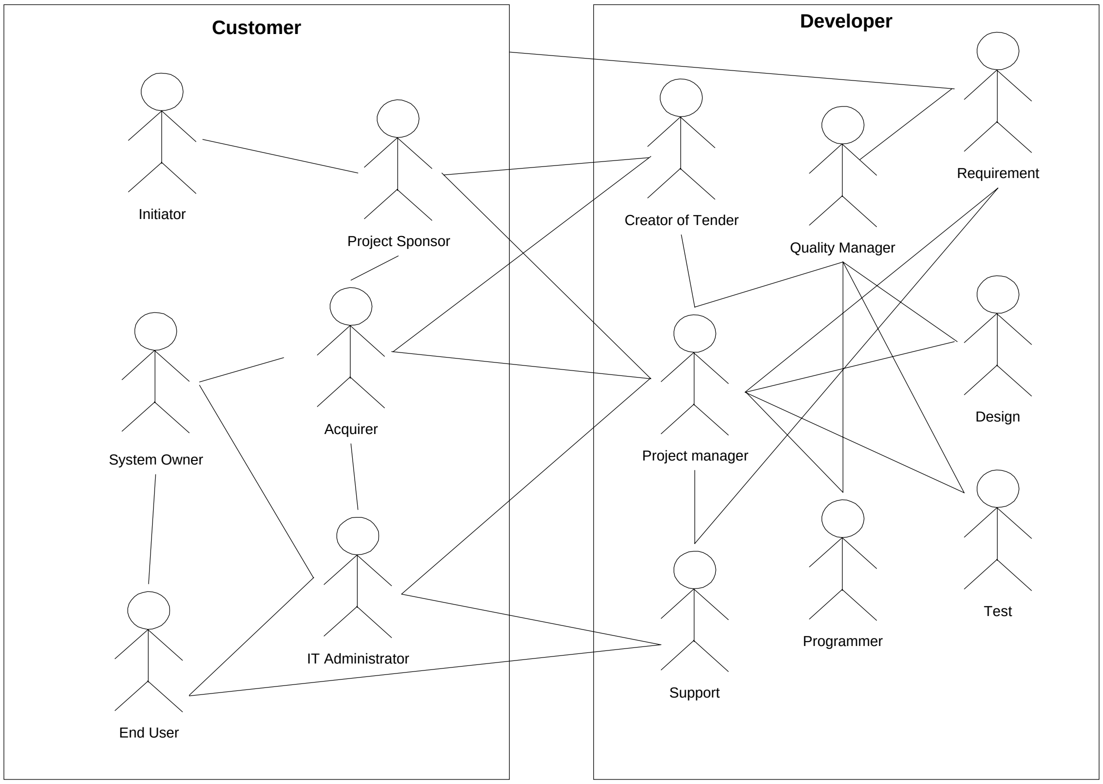


Figure 14: Actors and possible communications in a software development project.

The lines between the actors of Figure 14 represents communication in a possible development project. The space between the customer box and the developer box represents the customer developer interface. Some of the messages will be very precise and travel from an actor to another in a defined and verifiable form. Some messages will be more diffuse and perhaps reach the wrong person with modified content at wrong point of time.

3.3 Message Perspective

This section describes the customer-developer interface in the perspective of the messages passing between the customer and the developer during a software development project. Communication theory describes an information process in the terminology of Figure 15 below.

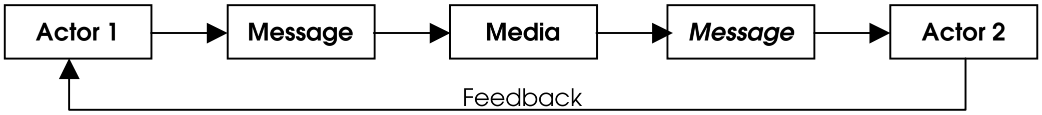


Figure 15: Basic communication process

Each connection line of Figure 14 can be said to represent such a communication process.

The right message box is written in *italics* because the message has been affected by the media. This means that the receiver never gets exactly what the sender has sent. The feedback arrow represents the same kind of *information process,* but in the opposite direction. If the feedback information can be related to the first message, the duplex information flow has turned in to *communication*.

In some contexts the term *media* is confused with the term *channel*. In this report, *media* describes by which technical form a message is transferred, e g by phone mail or live meeting. A *channel* is something that has a specific intellectual content, a defined sender and receiver and uses one or more specific media. Examples of channels are CNN, The Washington-Moscow Hot Line or a weekly news-mail from a project manager.

The three components of the communication process (actors, message and media) are excellent bases for deeper analysis of the basic component in the communication process – *the message*. A fourth aspect, related to the *state* of the receiving actor, will also be introduced. Together they will build a foundation for a fertile categorisation of the messages. The four aspects are:

- The Sending and Receiving Actors
- Message Contents
- The Media
- The Level of Surprise

3.3.1 The Sending and Receiving Actors

Each message can be defined by who is the source and who is the receiver. Section 3.2, describing the actors, gives a picture of which actors at one side that are directly involved in interaction with the actors on the other side or somehow connected to the interface. Remark the consequences of the restriction made in Figure 5: Single interface development at page 18. The study is restricted to one interface that might have to be used by several involved actors. A designer might request an clarification direct from an end-user or send the request via a project-manager at the developers side and a system owner at the customer side. Both these scenarios are legal here. This means that the possible number of combination is n*(n-1) bi-personal interfaces. Add the possible multi-personal links and the ability to broadcast - the number of unique connections is enormous.

3.3.2 Message Contents

Another straightforward way of defining a message is by its contents. In a software development context one can separate *product information* from *process information*. *Product information* deals with the functionality or features of the product in the form of functional and non-functional requirements. *Process information* deals with the organisation of the development project e.g. questions like when, where and who? Most of the messages can be assumed to deal with requirements. From the customer side the following possibilities are likely to occure:

- New requirements
- Demand for changes in existing requirements

- Removal of existing requirements

From the developers side, the following messages seems very likely:

- Request for clarification of a requirement
- Request for validation of a requirement
- Suggestion to change of a requirement

Concrete examples of process information are:

- Time, place and participants for putting the requirements in baseline
- Request for process status
- New deadlines
- Progress reviews
- External audits
- Time and amount for partial financial settlement

3.3.3 The Media

This section handles the third aspect of the messages - the form in which the information is distributed, the media. Is a new requirement communicated by an informal phone call or by an electronic request-for-change-form? Is a validation made by a baseline review meeting or over a lunch pizza? Every channel or media is combined with certain risks and certain advantages. To exemplify this, Figure 16 below lists some common communication media and some, but absolutely not all, of their qualities.

Media	**Advantage**	**Risk**
Telephone call	Full duplex Rapid	Hard to record and verify
E-mail	Rapid Available Storable	Technical Security problems
Letter	Sense of formality Hard to change Signature Static	Slow Only one version Static
Fax	Can communicate hand made illustrations and sketches	Gets lost in office
Web page	Available in office environment Updateable	Static structure
Formal meeting with agenda and record	Full (multi-) duplex for those present	Non-present persons might not be informed Time for attestation of record
Informal meeting	Full duplex Rapid Socially nice Creative	Hard to record and verify

Figure 16: Risks and advantages of communication media

The contents of Figure 16 are by no means complete. The purpose is to give an idea of questions that has to be dealt with when deciding on by which media a certain messages best should be communicated.

3.3.4 The Level of Surprise

The fourth approach in how to categorise the messages is inspired from a risk management perspective. How the situation in which the message is received or *the level of surprise* affects the reception of the information. Is the message part off a planned interaction or is it not? Is the message welcomed or feared? One can assume that a planned and positive message is received and understood in a better way than a negative message that comes as a complete surprise. Some possible categories are listed below:

- scheduled
- expected but not scheduled
- feared and analysed in risk analysis
- feared but not analysed in risk analysis
- a complete surprise

It is assumable that the reception of the messages in the upper categories will be both better and cheaper than the reception of the lower categories, independent of whether the customer or the developer is the sending part.

3.3.5 Conflict between process models and messges

One basic component of a process model (Section 3.1) is the *high-level activity*. Sending and receiving messages can be regarded as *low-level activities* and it is hard to draw an exact dividing-line between low-level and high-level activities. This problem becomes obvious when to decide if for instance a requirement workshop should be classified as an activity or as a message. Therefore: *meetings* that have a stringent focus at transfer of a defined type of information is regarded as *messages* in the report.

3.4 Summary

This section has outlined three perspectives on different levels of how a customer-developer interface can be described:

- **Process models:**
  How is the development process organised on a high level and what consequences will the chosen organisation have on the customer-developer interface?
- **Actors and stakeholders:**
  Which actors are involved in the development process at respective site and how do their roles and stakes affect the customer-developer interface?
- **Messages:**
  Who communicates what *message* to *whom,* by which *media* and under which circumstances?

Each of the three perspectives captures some unique features, phenomenon and rules that can be assumed to affect the customer-developer interface. A description of a customer-developer interface based on process models, actors and stakeholders and messages should make it possible to analyse the pros, cons and development areas of the customer-developer interface in a comparable way.

4 The Customer-Developer Interface According to Best Practices of Software Development

A lot of well-written standards, process models, reference models, best practices, scientific articles and management guidebooks focus on explaining how to develop software have published. Some examples are:

- Capability Maturity Model – CMM [SW-CMM]
- Capability Maturity Model Integrated – CMMI [CMMI 1.02d]
- Software Process Improvement and Capability dEtermination – SPICE [SPICE]
- The Institute of Electrical and Electronics Engineers – IEEE [IEEE std 1062-1993]
- Extreme programming – XP [Beck 1999]
- Rational Unified Process – RUP [Kruchten 1999]
- Dynamic Systems Development Method – DSDM [Stapleton 1997]
- Department of Defence (U.S.A) - DoD

Many of these standards and models also deal with the consequences of splitting the development process on two organisations, and thereby they also deal with the customer-developer interface. Some of the models give clear recommendations of how and when to interact with the customer. Some models have only little to say on the issue.

The objective of this section is to describe some models and best practices; most of them focused on software development. From the descriptions will then *objectives* for proper customer-developer interfaces and *practices* for reaching these objectives, be extracted

In order to avoid confusion and mix up with the theoretical perspective of *process models* from Section 3.1, the standards, models and practices from now on will be termed *best practices.* They will be called so regardless of whether the originators of the best practice consider them to be best practices or not. To clarify this in short;

> *Objectives and practices for the customer-developer interface are to be extracted from best practices of software development*.

Most of the chosen best practices focus processes and activities that happen *inside the development organisation.* There is no absolute secure way of knowing if these best practices in combination really cover all relevant objectives and practices of *the customer-developer interface*. One way of determining if they do, is to investigate if the best practices describe similar objectives and practices or if they are mutually very disparate. The arguments are as follows.

If, on one hand, all the best practices identify the same problems and describe similar practices, it can be assumed that the problem is relevant and that the described practice is good and useful. If, on the other hand, the objectives and practices are disparate, new variants of problems and practices have been identified and can be added to a list of qualitative variations. The way to know when to stop is to make a trade-off between the cost of investigating new best practices and the probability to find not yet discovered objectives or practices.

There is a risk that identified similarities between the best practices is due to that one best practice is an imitation of the other. A cure for that risk is to choose models from different sources, with different perspectives or even with competing driving forces.

The choice for which best practices to examine is, by other words, based on the assumption that the learning that can be made from each of them and from their attitude towards the customer-developer interface, to some extent complements the learning that can be made from the others. In the terminology of qualitative methodology this is the same that the researcher wants *to maximise the coverage of the qualitative variation* (see Section 2.2, page 25).

The trade-of between the risk of missing objectives and practices on one hand and the cost of investigating more best practices on the other hand ends in this study with the choice of the following best practices:

- SEI Software Acquisition Capability Maturity Model – SA CMM [SA-CMM1.01]
- SEI Team Risk Management – SEI TRM [CMU/SEI-94-SR-5]
- The Gaps-model [Zeithamal 1996]
- Rational Unified Process – RUP [Kruchten 1999]
- Extreme Programming – XP [Beck 1999]
- Dynamic Systems Development Method – DSDM

The structure of the section is as follows. Each best practice is first described in general. After that follows a description of how the model looks when captured by each of the three theoretical perspectives outlined in Section 3. Each model is finally analysed regarding:

1. To what extent the theoretical perspectives could capture the best practice.
2. If any new objectives and practices of the customer-developer interface were revealed by the best practice.

4.1 Capability Maturity Model

The Software Engineering Institute (SEI) of the Carnegie Mellon University has developed the CMM concept of software process improvement during the late eighties and nineties. The major sponsor has been the American Department of Defence (DoD). The main argument for this was the need to get control over cost and quality of the software products delivered by its contractors. The intellectual framework was founded in the eighties and early nineties by Watts Humphrey in work like “A Discipline for Software Engineering” [Humphrey 1995].

One of the strengths of CMM is that it is based on contributions from software developing practitioners. The late versions of Software-CMM [Paulk 1997] and Software Acquisition CMM [SA-CMM 1.01] has been validated and proven by use in numerous organisations and development projects. This indicates that, what is recommended by these documents is really relevant.

The interface between the customer and the developer are treated in many different SEI documents. *Software Acquisition CMM* [SA-CMM 1.01] was designed to guide the acquiring part to gain maturity and *Software CMM* [Paulk 1997] was designed to guide the developer. Lately, these to models and a third model, System Engineering CMM, has been integrated to CMMI (Capability Maturity Model – Integrated) [CMMI 1.02d]. Another concept designed by SEI in order to deal with the customer-developer interface is Team Risk Management. Team Risk Management will be presented in Section 4.2.

Section 4.1.1 describes features and structures that are common for of all of the CMM’s. The Sections 4.1.2 to 4.1.6 penetrates the customer-developer interface as described in *Software Acquisition CMM*.

4.1.1 Generic Description of CMM

The generic description of CMM in this section is primary based on texts about SW-CMM 1.1. SEI defines two major areas for using the model [Paulk 1993, p 43]:

- To assess the current development process used by an organisation in order to improve its ability to deliver high quality software. This is what a CMM-climbing organisation does in order to compete better on the market of software.

- To evaluate a development process used by a potential sub-contractor in order to gain trust in its ability to deliver high quality software. This is what the American Department of Defence wants to do when selecting software contractors to military systems.

A third user domain is to be inspired by the huge work invested in building and validating the model and derive lightweight process models from it.

CMM structure
The keywords of CMM are maturity and capability. The basic assumption permeating the Capability Maturity Model is that software development organisations with a structured way of doing their job will over time deliver software with higher quality than less well structured organisation will. In CMM terminology this means that a well-structured organisation has better capabilities to deliver high quality software. This capability is the “C” in “CMM”.

For everybody who agrees on the assumption above, CMM offers a concept of:

- How an organisation can get well structured.
- Where to start and how to control the organisation evolution process.

By CMM terminology this would be expressed: how an organisation might gain maturity, Maturity is the first “M”. The concept or *m*odel stands for the second “M”. *Maturity* and *capabilities* are very central to the model.

Immaturity
An immature development organisation is likely to be characterised by statements as:

- The development processes are genererally improviced.
- The project management is focused on solving immediate crises.
- Functionality and quality of the product often have to be reduced in order to meet deadlines.
- The organisation has no objective ways to judge quality

Maturity
In contrast a very mature organisation may be characterised by these statements:

- The organisation has an organisation-wide ability to manage software development and maintenance.
- Development activities are carried out according to a defined and by all parties understood and recognised development process.
- These processes are gradually improved and updated.
- The roles and responsibilities within the organisation are clear and understood by its participants.
- Management monitors the quality of the product and the process.
- The organisation keeps an objective base for a quantitative measure of quality.

If a software development organisation wants to transform from an immature to a mature organisation is the SEI Capability Maturity Model one of several, but heavily tested way to go.

The software process capability is the range of expected results that can be achieved and thereby a way of predicting the most likely outcome of the next project. The five *maturity levels*, which represent the steps to climb in order to reach the organisational maturity, are:

1. Initial
2. Repeatable
3. Defined
4. Managed
5. Optimising

Next central concept of the CMM is the Key Process Area. The *Key Process Areas* are the building block of each capability level.

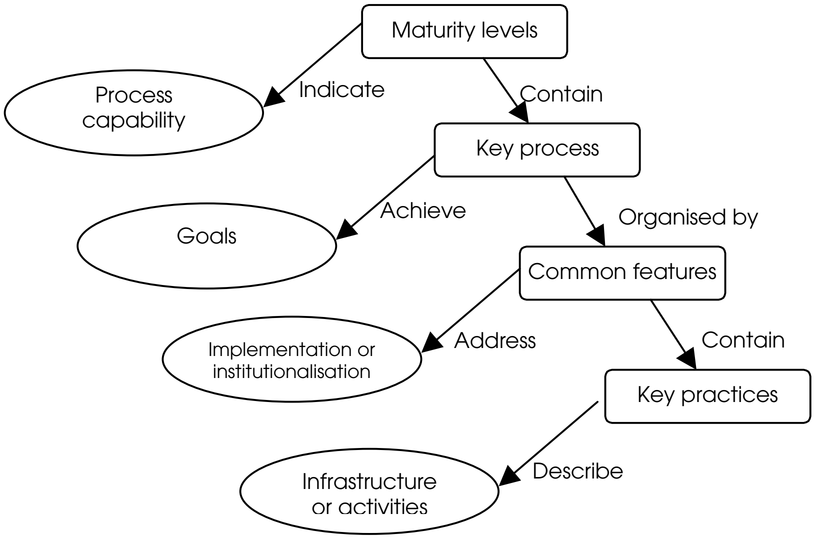


Figure 17: CMM structure - Goals, Common features and Key Practices

Figure 17 pictures the Key Process Area in its CMM context. Each Key Process Area is defined by its *Goals*, *Common Features* and *Key Practices*. The goals are very concrete. An example from Key Process Area: *Software Configuration Management* at level 2 is in short [Paulk 1997, p 180] that:

- Configuration management is planned.
- The configuration-managed products are controlled, identified and available.
- Changes to the products or deliverables are controlled.
- Relevant people have access to status information about the software and deliverables.

Each Key Process Area is also described according to five common features listed below together with explaining questions:

1. **Commitment to perform:** Do we have a good reason to become better?

2. **Ability to perform:** What organisational features have to be established before the project for improvement of the Key Process Area starts off?

3. **Activities performed:** How shall we do the job when we do it better then before?

4. **Measurements and analysis:** How can we measure to decide whether we do what we planned to do?

5. **Verifying implementation:** Did we do what we planed to do?

4.1.2 The Customer-Developer Interface According to Software Acquisition CMM

Software Acquisition CMM [SA-CMM 1.01] was designed to guide the acquiring part to gain maturity in its role. What recommendation has Software Acquisition

CMM to make about the customer-developer interface? The material to investigate is very large. In order to get out the most relevant materiel the following condensation is performed. Figure 18 lists the key process areas of SA-CMM.[5]

Level	Focus	Key Process Areas
5 Optimizing	*Continuous process improvement*	• Acquisition Innovation Management • Continuous Process Improvement
4 Quantitative	*Quantitative management*	• Quantitative Acquisition Management • Quantitative Process Management
3 Defined	*Process Standard-isation*	• Training Program • Acquisition Risk Management • **Contract Performance Management** • Project Performance Management • Process Definition and Maintenance
2 Repeatable	*Basic project management*	• **Transition to Support** • **Evaluation** • **Contract Tracking and Oversight** • Project Management • **Requirements Development and Management** • Solicitation • Software Acquisition Planning
1 Initial	*Competent people and heroics*	

Figure 18: Key Process Areas of Software Acquisition CMM

In SA-CMM, all key process areas, in some way or another, deal with the customer-developer interface. The best description of them is made in the original document [SA-CMM 1.01, p 17-112]. Since this study focus on the ad-hoc customer, there is a limited scope for process improvement. Therefore an analysis can be restricted to the key process areas, goals and practices of the maturity level two and of the maturity level three. Some of the key process areas at level two and three focus on aspects that are private to the customer. Left, dealing with the interface, is the key process areas, marked with bold style in Figure 18. Figure 19 below lists them together with the top-level activities that can be assumed to affect customer-developer interface.

[5] Remark that the structure of SA-CMM is the same as the structure of SW-CMM. The difference between them lies primarily in the purpose, the user and the Key Process Areas.

Maturity Level	Key Process Area	Top-level activities
2	Requirement development and Management	5. Bi-directional traceabilty between the software-related contractual requirements and the contractor's software work products and services is maintained throughout the effort.
	Contract Tracking	2. The project team reviews required contractor software planning documents which, when satisfactory, are used to oversee the contractor's software engineering effort. 3. The project team conducts periodic reviews and interchanges with the contractor.
	Evaluation	3. The evaluation requirements are incorporated into the solicitation package and resulting contract 4. The project team assesses contractor's performance for compliance with evaluation requirements. 6. Results of evaluations are analysed and compared to the contractor's requirements to establish an objective basis to support the decision to accept the product and services or take further action.
	Transition to Support	2. Responsibility for the software products is transferred only after the software support organisation demonstrates its capability to modify and support the software products.
3	Contract Performance Management	3. The contractor's software engineering process is appraised according to the project's defined software acquisition process. 4. Results of the contractor's engineering activities are appraised according to the project's defined software acquisition. 5. Measurements are used to apprise the contractor's performance and trends analysed. 6. As understanding of the software engineering process, products, and services improves, the project team may propose changes to the software products or services, process descriptions, plans and activities. 7. The end user periodically participates in the evaluation of evolving software products and services to determine the satisfaction of operational requirements. 8. Contract performance management activities are performed to foster a co-operative environment between the project team and the contractor.

Figure 19 Key Process Areas and Top-Level-Activities dealing with customer-developer interface in SA-CMM level two and level three (SA-CMM 1.01, Appendix C)[6]

Keeping in mind that SA-CMM says a lot about *what* is to be done and little about *how* it is to be done, SA-CMM gives more detailed recommendations than that of Figure 19. But without penetrating the top-level activities any further, the content of the left column of Figure 19 highlights practices of the customer-developer interface that are important in the perspective of SA-CMM.

4.1.3 Process model of SA CMM

CMM gives no opinion whether a waterfall model or incremental model is to be used. But the list of activities in Figure 19 can be seen as the list of important top-level activities at the customers side that should be performed, without deciding when, how

[6] The number references the position in the original document.

often and in what order. Suppose the development was to follow a waterfall process model, then Figure 20 is a fragmentarily illustration of the high level interface according to SA-CMM describing high-level activities at the customer side mostly aiming for controlling the process and product at the developers side.

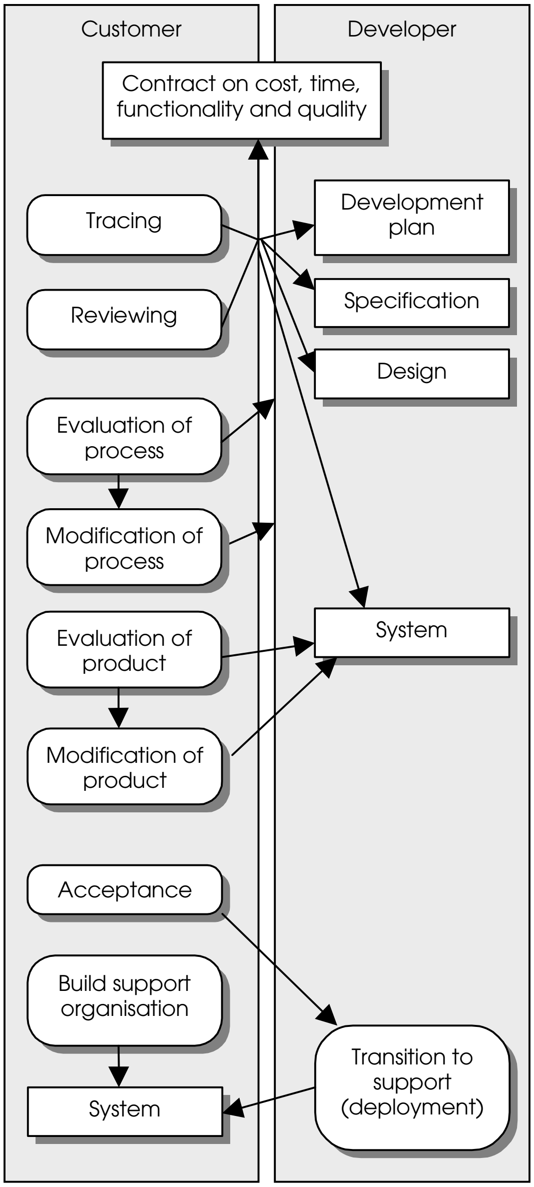


Figure 20: Process model perspective of the customer-developer interface according to Acquisition CMM

4.1.4 Actors and stakeholders of SA-CMM

The actors and their roles that can be identified in the top-level activities of Figure 19 is, in short, listed in Figure 21 below.

Actor	Roll
Customer Project Team	Maintains bi-directional traceabilty between requirements and software product. Reviews developer's software plan. Communicates with the developer. Assesses developer performance. Adjust the developer's development process. Adjust the software product.
Customer Support	Receives responsibility for the software on deployment
Customer End User	Participates in evaluation of evolving software product and services
Developer	Plans the software development. Maintains bi-directional tractability between requirements and software product. Develops the software product. Transfer the product to the customer

Figure 21: Actors and their roles according to SA-CMM

4.1.5 Messages of SA-CMM

Messages that can be outlined in the condensed description of Figure 19 are listed in Figure 22 below.

Product information	Source	To	When	Media
Contract	Customer Developer		---	---
Requirements	Customer	Developer		
Software product	Developer	Customer		
Acceptance	Customer	Developer		
Process information:				
Contract	Customer Developer			
Project plan	Developer	Customer		
Project audits	Customer Developer			

Figure 22: Messages according to SA-CMM

Since CMM says little of how things are to be done, it is consequent that the model has no suggestions regarding timing and media for messages

4.1.6 Analysis and Summary

The choice of describing the SA-CMM instead of other CMMs automatically puts the focus to the responsibilities of the customer. The process model perspective revealed some customer high-level activities relevant for the customer-developer interface. In the SA-CMM the main customer responsibility is to monitor and control the activities and deliverables of the developer in order to get confidence in the software under development. The developer develops the system and serves the customer with product and process information.

An objective for the customer-developer interface highlighted by SA-CMM is the importance of the customer getting confidence in the development process and in the software product. The method to achieve the objective is to monitor and control the process and product. If the capability to perform the monitoring and control is low, then the developer’s responsibility to serve in these matters is increased in correspondence to that.

Regarding the *actors* and the *messages,* the theoretical perspectives helped to capture a picture of the customer-developer interface, but on a very high level.

4.2 The Customer-Developer Interface According to SEI Team Risk Management

From the descriptions of SA-CMM it might be close to that SEI only focus the responsibility of the acquiring part. This is not true. In order to deal with the problems of the communication between customer and developer SEI has introduced the concept of Team Risk Management. In short it recommends an introduction of a joint forum for risk management. This is in fact a new organisation that can be said to be *The Interface*. The concept is best introduced by two quotations:

> “Team Risk Management is a new paradigm for managing programs or projects by developing a shared product vision, focused on results, and using the principles and tools of risk management to cooperatively manage risks and opportunities.“ [CMU/SEI-94-SR-5, p 1]

Team Risk Management is an extension of basic Risk Management and to fully understand the *Team Risk Management* one has to be familiar with SEI’s basic Risk Management. The SEI definition of Risk Management:

> “Risk Management sets forth a discipline and environment of proactive decisions and actions to
> 1. assess continuously what can go wrong (risks).
> 2. determine what risks are important to deal with.
> 3. implement strategies to deal with those risks.” [CMU/SEI-94-SR-5, p 6]

Traditionally Risk Management is an activity that is initiated during project planning and then sustained and evolved during the project. The objective is to identify risks and then, in best case, avoid them or otherwise reduce the negative impact the risks might have on the process or on the product. The concept of *Team Risk Management* extends the *Risk Management* by moving the responsibility from the project manager at the developer’s side to a team consisting of representatives from both the customer and developer organisations. Even if the focus of this joint forum is on *risks,* the concept has several benefits. First, many of the risks that will come up on the risk management agenda will probably deal with problems related to the customer-developer interface, leading to prevention actions. Second, working together will bring about a better understanding of the application domain respective restrictions enforced by the nature of software development. Third, the Team Risk Management activity has a chance to bring about a *shared product vision* between the parties. Both organisations will synchronise the image of the product under development and become better to make good decisions.

4.2.1 Process Model of Team Risk Management

Figure 23 illustrates the basic activities of Team Risk Management. The shadowed boxes illustrate high level activities. The inner white boxes illustrate activities of risk management at a lower level. Team Risk Management does not deal directly with the basic engineering aspects of the development process, e. g. design and test as described in Section 3.1. It can rather be described as an activity focused on supporting the continuos planning of the engineering activities.

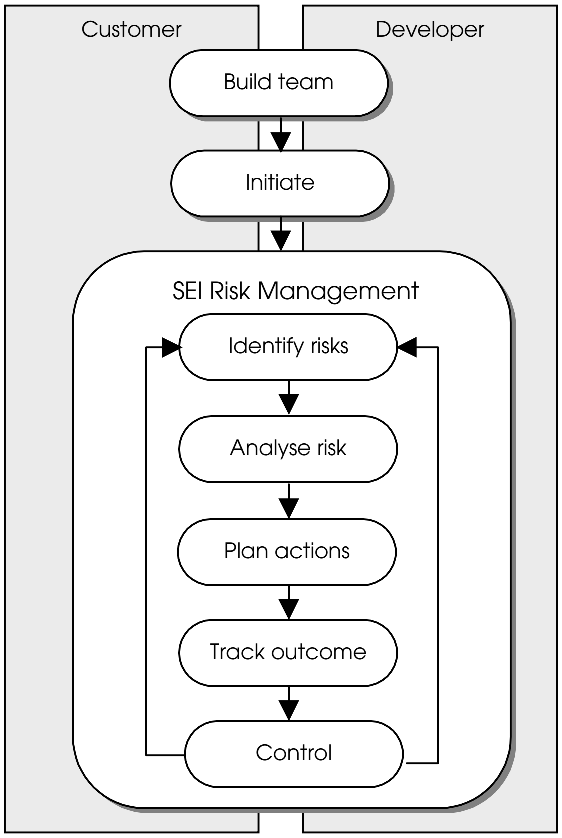


Figure 23: High level activities of SEI Team Risk Management (CMU/SEI-94-SR-5, p 13)

4.2.2 Actors in Team Risk Management

The concept of Team Risk Management deals with the actor categories:

- Customer
- Developer
- The team: the customer and developer together.

4.2.3 Messages in Team Risk Management

From studying [CMU/SEI-94-SR-5] it is possible to extract some messages, which are listed in Figure 24 below. Since the risk management activities are performed in the joint organisation it is hard to define a source and a receiver for the messages.

Product information	**Source**	**To**	**When**	**Media**
List of risk regarding product	Customer Developer		Identification	
Process information:				
Team Risk Management Initiative			At start up	
List of risk regarding process			Identification	
Risk mitigation plans			Analyse	
Action plans			Plan	
Meeting				
Team review				

Figure 24: Messages in Team Risk Management

4.2.4 Analysis and Summary

The process model perspective of Team Risk Management introduced interface related high level activities that was not mentioned among the engineering activities in Section 3.1 – the joint Risk Management. A new concept introduced by Team Risk Management, mostly related to the process model perspective of the customer-developer interface, was the bridging accomplished by introducing a new and common organisation – the interface itself. This aspect was also captured by the actor-perspective in the form of the *joint organisation*.

The message-perspective introduced a lot of documents regarding Risk Management. One objective of Team Risk Management is to facilitate opportunities for the customer and the developer to better understand the problem domain of the other part. During the process of achieving that, the list of risks that relate to the product will grow continuously, describing the most critical aspects of the development. At the same time there will be a continuos transformation of domain knowledge from the customer side to the developer and from the developer to the customer. This is aspect illustrated by *the list of risks* in Figure 22.

One of the key objectives of Team Risk Management is to generate a shared vision or a team spirit. This objective was not captured by any of the three perspectives.

4.3 The Gap-model of Service Quality

Software engineering is not the only discipline dealing with customer-developer communication. As an instrument in the theory of service marketing, *the gaps-model of service quality* [Zeithamal 1996] illustrates how a supplier can minimise the gap between the customers' *expectations* of a service and the customer's and perceptions of the received service:

> “In a perfect world, expectations and perceptions would be identical: customers would perceive that they receive what they thought they would and should. In practice these concepts are often, even usually, separated by some distance. Broadly it is the goal of services marketing to bridge this distance…” [Zeithamal 1996, p 38]

Adapted to the word of software development the Gaps-model aims at reducing the distance between the customer’s (initial) expectations or outline vision of a software product and the customer’s image of the result. One reason for why the Gaps-model can be interesting from a perspective of software development is that it can be said to deal with the fundamental problem of transfer and transformation of knowledge from the application domain to the software engineering domain. The object of the model is the *conceptual gap*. The objective is to reduce and even eliminate that gap. The strategy is to break it into manageable sections that can be dealt with one by one.

Remark! The conceptual parallelisms between *marketing and providing of services* on one hand and *creative development of software* on the other hand are not total - but big enough to motivate the spending of time and printing ink on the model.

This section will briefly describe the Gaps-model and how it may affect the customer-developer interface. The basic axiom for the Gaps-model is that the objective for at service provider is to minimise the gap between customer’s expectations of a service and the same customer’s perception of the perceived service - in other words to reduce gap 5 of Figure 25 below.

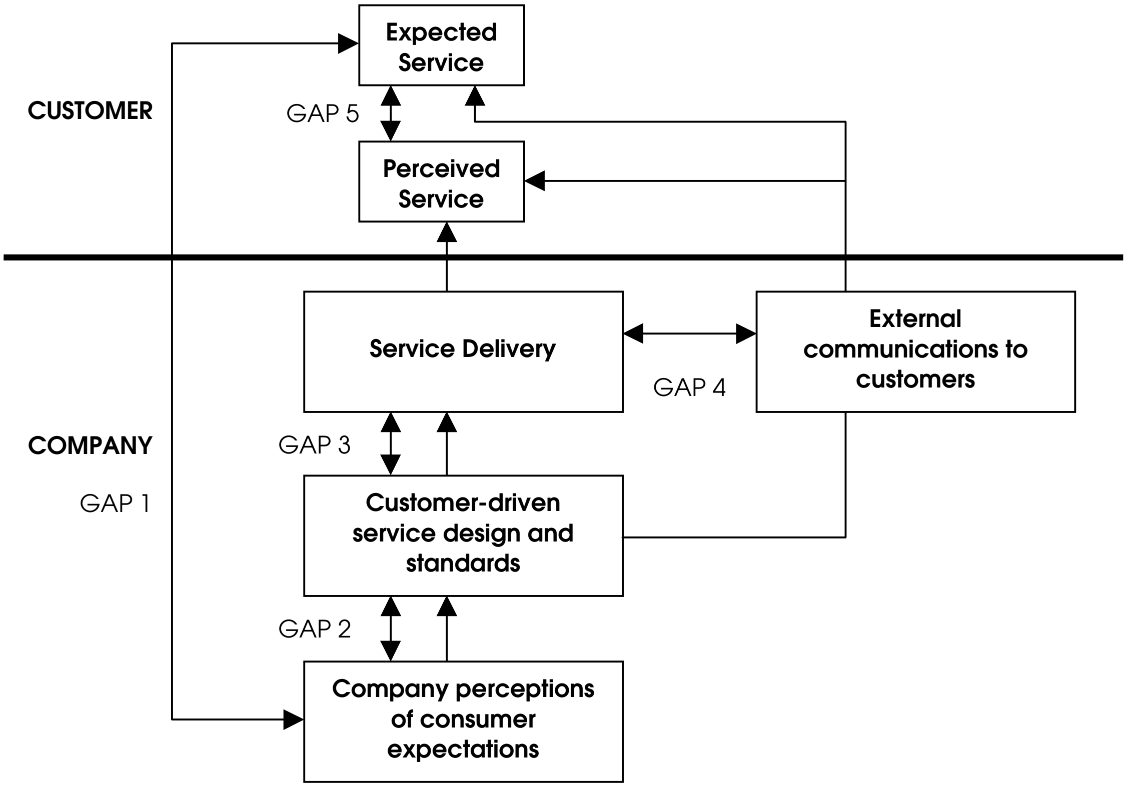

Expected Service
CUSTOMER
GAP 5
Perceived Service
Service Delivery
External communications to customers
GAP 4
COMPANY
GAP 3
GAP 1
Customer-driven service design and standards
GAP 2
Company perceptions of consumer expectations

Figure 25: gaps model of service quality (Zeithamal96, p 48)

The strategy for closing the gap at the customer's side lies in closing four different gaps at the company's side (gap 1-4 of Figure 25). Each of these gaps has some certain characteristics, some likely explanations and can be handled by some recommended actions in order to shrink the gap.
Below the gaps are described translated into a software development terminology:

Gap 1 - expected service vs. company perception
Gap 1 describes the difference between the customer expectations of what is to be developed and how the developer perceives these expectations. Possible reasons for the gap may be that there is no direct interaction with the customer, the developer is for some reason unwilling to ask for the requirements or incapable of addressing them.

The main action recommended when marketing services is to establish e. g. market surveys, complaint systems or customer panels. Translated to a software engineering activity this could be translated to structured requirement elicitation through methods like interviews, user observation, focus groups etc.

Gap 2 - Company perception of expectation vs. Service design and standards
Gap 2 describes the problems that might occur if the developer is fully aware of the customer expectations but unable to meet them due to organisational practices, routines or standards that obstruct the adoption of the system under development to the customer expectations. An example in software engineering may be eagerness to

re-use components in favour of flexible adoption. A less concrete reason for the deviation may be as simple as lack of commitment in the developing organisation.

To reduce and eliminate gap 2, Zeithamal et. al., among other things, recommend the use of barometers and surveys of customer satisfaction. From a software engineering perspective this could be translated to applying usability engineering methodology and evaluate usability metrics.

Gap 3: Service design and standards vs. Service delivery

Gap 3 is the gap that occurs if the developer is fully aware of the customer expectations and at the same time capable of developing and delivering the right product but yet fails to do so – mainly due to human resource factors. According to the Gaps-model the reasons for this failure is to be found in factors like:

- The involved personnel does not understand their role
- The employees feel in conflict between customers and the developer management
- Inadequate technology
- Lack of empowerment

In software development gap 3 is likely to be widened by if for instance the developer organisation forces a project to follow inadequate development processes or to use a certain CASE-tools. Another problem that could expand gap 3 is a classic example where development process or product metrics stimulates writing a lot of code lines in favour of building compact and maintainable systems.

Actions in order to reduce gap 3 could for instance be to empower the development team in order to aim for the customer needs rather than to satisfy developer-internal policies.

Gap 4: Service Delivery vs. External customer communication

Gap 4 is the gap between what the developer says it delivers and what the customer believes it will get, with focus on the saying part. If the developer initially promises features that will not be fulfilled, the gap 4 will be widened. Another aspect that might open the gap is when the developer fails to show or convince the customer that the initial promises actually have been fulfilled.

Recommended action is to try to influence the customers' image of the system before development, during development as well as on deployment

4.3.1 Process Models of the Gaps-model

Expressed in the process model syntax, the Gaps-model in Figure 25 above, in short, will look like Figure 26 below.

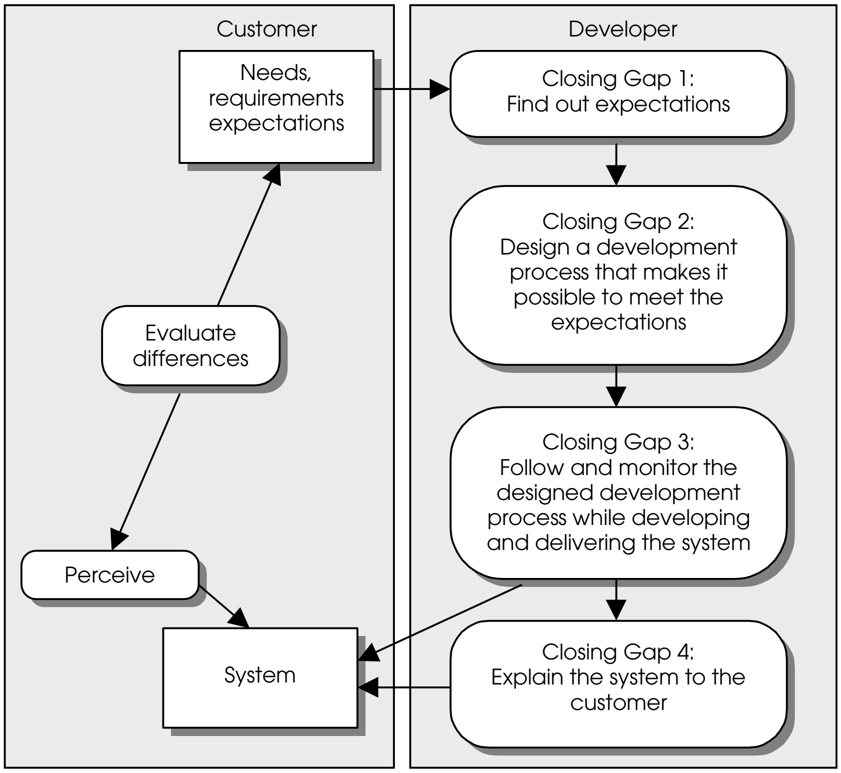

Customer
Developer
Needs, requirements expectations
Closing Gap 1: Find out expectations
Closing Gap 2: Design a development process that makes it possible to meet the expectations
Evaluate differences
Closing Gap 3: Follow and monitor the designed development process while developing and delivering the system
Perceive
System
Closing Gap 4: Explain the system to the customer

Figure 26: Process model of the Gaps-model

The Gaps-model has very little and, at the same time, very much to say about top-level activities of software development and how to arrange the activities over time. Problems and recommended actions introduced by gap 1 to 3 are rather to be used as a qualitative input when designing the high-level engineering activities independent of whether the project follows a waterfall or an incremental development process.

4.3.2 Actors and Stakeholders of the Gaps-model

The Gaps-model does not fit the actor and stakeholder perspectives too well either. At a high level there is the *customer* and the *developer*. At a low level there is an enormous amount of different actors and players overlapping each other in different organisations. Some of them are more essential and some of them are more peripheral to the activities of software development.

4.3.3 Messages of the Gaps-model

Both the Figure 25 and Figure 26 indicate three high-level messages listed in Figure 27 below.

Product information	**Source**	**To**	**When**	**Media**
Needs, requirements expectations	Customer	Developer	---	---
Developed system	Developer	Customer		
System explanation	Developer	Customer		
Process information:				

Figure 27: Messages in the Gaps-model

4.3.4 Analysis and Summary

The process model-perspective introduced a new perspective *of the objective of the traditional high level engineering activities* – where to look for the conceptual gap between the customer expectations and the delivered system. The process model perspective also introduced a new high-level activity for the later part of the development process. In order to facilitate the customer's ability to compare initial expectations with the delivered product, the system has to be explained to the customer. This explanation should be expressed in the same form and language as the customer's initial vision. Compared to the development activities of Section 3.1, the activities of the Gaps-model are best regarded at some meta-level activity or as low-level activities in the project planning process. Another target activity that ought to be influenced by the Gaps-model is the developer's aspects of validation.

From the marketing domain it is to learn the need for specialist competence in how to understand the customer expectations and how to learn the business rules of the application domain. Another learning is that it could be a good idea to put some energy into telling the customer what he/she actually has acquired, in a form and language that matches the form and language of the initial needs, requirements and expectations.

The actor perspective of the Gaps-model did not prove to be very useful but the message perspective helped in finding a new and very useful message – the *system explanation*.

One thing the three perspectives could not capture was the conceptual gap itself, but the Gaps-model indicates a possible way to break down the gap in manageable parts.

The Gaps-model carries a problem in its assumption that the requirements, needs and expectations of the customer are *static* throughout the entire development. That assumption will probably never be fulfilled in a software project with an inexperienced ad-hoc customer.

4.4 The Rational Unified Process – RUP

The Rational Unified Process is a defined and detailed software development process available for customers of Rational Software Corporation[7].

“The goal of the process is to produce, within a predictable schedule and budget, high-quality software that meets the needs of its users.” [Kruchten 1999, p ix] and in that sense the RUP does not differ much from other best practices. Worth noticing is the mentioning of *the needs of the end users*. RUP is truly aware of the possible complications on the customer side of the development processes.

The Rational Unified Process deals with the customer-developer interface in several different ways. In order to describe these, the basic structure of the process model and some terminology has to be introduced. The analysis in this section is based on [Kruchten 99] and the RUP version RUP2000, which is a 50 megabyte 2000 file large web-package of interrelated UML[8]-charts, specifications, definitions etc.

RUP can be described as a mix of waterfall and iterative development. The process is iterative in the short run. But since the emphasised activities of each iteration gradually changes through different phases in the long run of the development process it also has some waterfall characteristics. The iterative approach is very central to RUP.

The process is designed to be used together with advanced CASE tools. Therefore another core concept of the process model – *the artefacts* – are made possible. The artefacts constitute the software to be developed and its surrounding elements like for instance use-cases, descriptions of actors, architecture etc. From the artefact it is possible to extract *reports*. Reports are momentary images of the artefact that for instance could be used for letting the customer validate a set of specifications. The *artefacts* are objects for configuration and change management. The *reports* are not.

[7] "Rational Software Corporation (Nasdaq: RATL), the e-development company, helps organisations develop and deploy software for e-business, infrastructure, devices and embedded systems through a combination of tools, services and software engineering best practices. "(Rational)

[8] UML: Unified Modelling Language (Stevens 2000)

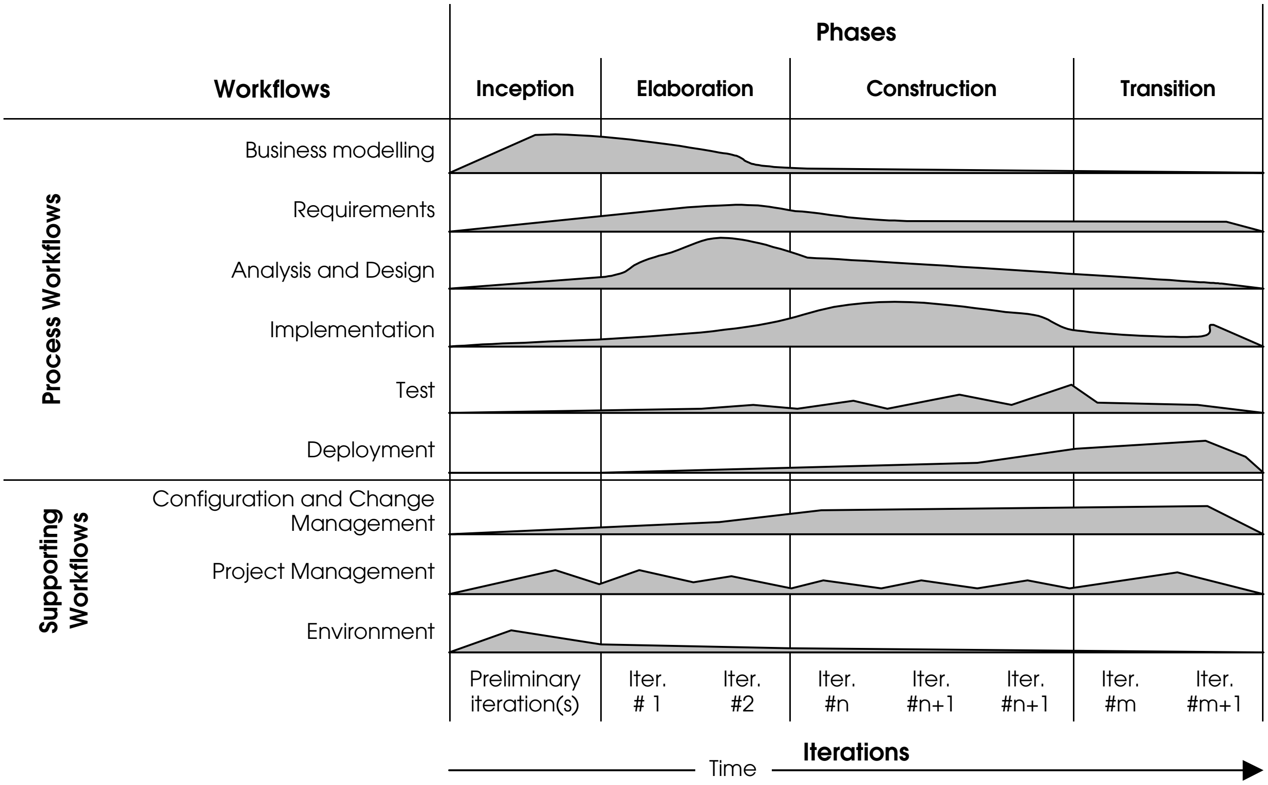


Figure 28: Rational Unified Process - Phases, Iterations and core workflows (Kruchten 1999, p 45)

Figure 28 above illustrates some basic terminology of the RUP. Each of the iterations consists of some or all of totally nine different workflows. The involved actors, concrete activities and the dependencies between the activities describe the workflows. The activities are described and argued for on very low level. There are checklists, vocabularies, explanations and hints about risks and their consequences. In other words who should do what, when and how? Each iteration, except from the first, is supposed to end up with a product that to some extent is executable and thereby usable for validation.

The goal of the inception phase is to specify the objectives of the project. The goal of the elaboration phase is to specify architecture and high level design. During the construction phase most of the software actually is constructed while the transition phase is dominated by deployment. In Figure 28 above, the shift in emphasis throughout the project, is illustrated by the shaded area, corresponding to each workflow. Between each of the four phases there is a formal milestone.

As indicated before, in RUP the interface between the customer and developer is covered in several different ways. And to get an answer to how RUP treats the interface, one has to look in several different parts of the process:

- The workflows:
  - Business modelling
  - Requirement
  - Deployment

- The concept of iterations itself
- The UML-syntax itself

These aspects will be dealt with in the following sections.

4.4.1 Process Model of the Rational Unified Process

From the perspective of the waterfall-evolutionary axis pictured in Figure 12, RUP can be described as a hybrid. The process is iterative in a short perspective. But since the emphasised activities of each iteration changes gradually through different phases in the long run of the development project, the process also has some waterfall characteristics.

Since all nine workflows, at least to some extent, are present in each iteration, the concept of iterative development means that there will be several formalised ways of communication between the customer and developer. In fact, one of the key arguments for the RUP is that the iterative approach bears the solutions to several customer related problems:

> “1. Serious misunderstandings are made evident early in the life cycle, when it’s possible to react to them.
> 2. This approach enables and encourages user feedback so as to elicit the system’s real requirements.
> …
> 8. Stakeholders in the project can be given concrete evidence of the project’s status throughout the life cycle.” [Kruchten 99, p 8]

The model prescribes, to a quite detailed level, when some instance of the customer is to be contacted on which issue and what to do with the answer. Workflows that involves the customer to a high degree are the following three:

- The *workflow: business modelling*[9] is primarily oriented towards creating and validating the developer’s side understanding of the rules, actors, activities and product flows of the customer business domain.
- In the *workflow: requirements*, actual requirements are elicited from end users as input to use-cases etc.
- The *workflow: deployment* deals with the introduction of the developed product at the customer site and thereby also with validation of functionality and quality.

Figure 29 below shows one iteration of the nine core workflows as top-level activities and which of the workflows that involves communication with the customer.

[9] The syntax means that *business modelling* is an instance of a workflow.

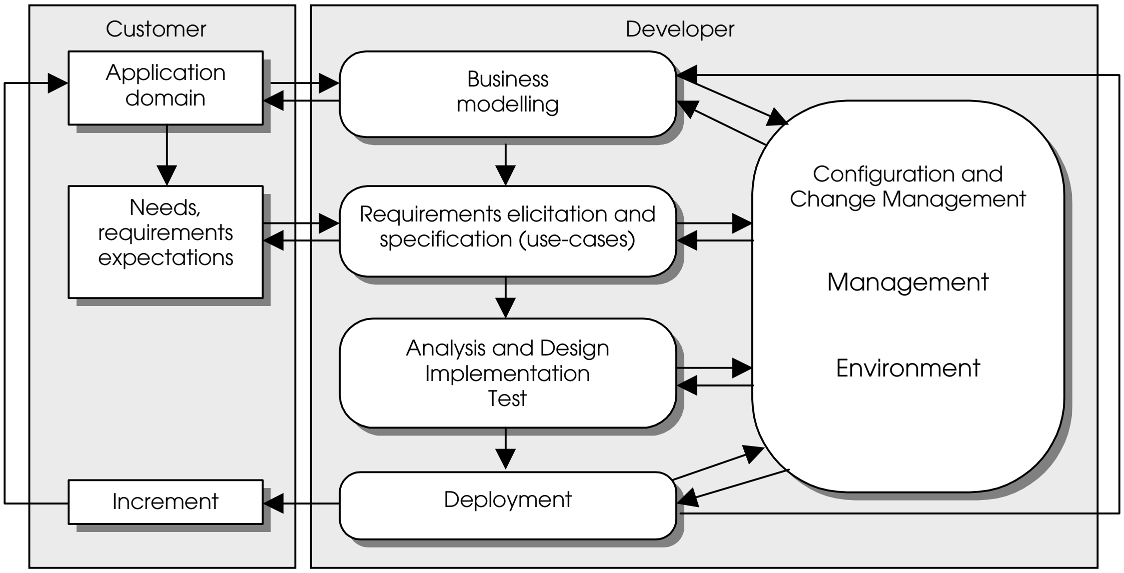

Customer
Application domain
Needs, requirements expectations
Increment
Developer
Business modelling
Requirements elicitation and specification (use-cases)
Analysis and Design Implementation Test
Deployment
Configuration and Change Management
Management
Environment

Figure 29: Process model of Rational Unified Process

Business modelling, requirements and deployment are the three cor workflows that includes interaction with the customer. Activities in the supporting workflows in the right box has only indirect contact to the customers

4.4.2 Actors and Stakeholders in the Rational Unified Process

In the activity charts of the process flows *Business modelling*, *Requirements* and *Deployment* the customer is mentioned as an actor involved in the activity.
In some of these activity charts the *actor: customer* are mentioned, in some others the *actor: stakeholder* and in yet other the *actor: end user* are mentioned. The *actor: stakeholder* is specified in Figure 30 below.

Actor: Stakeholder
A stakeholder is defined as anyone who is materially affected by the outcome of the project.

Effectively solving any complex problem involves satisfying the needs of a diverse group of stakeholders. Stakeholders will typically have different perspectives on the problem, and different needs that must be addressed by the solution. Many stakeholders are users of the system. Many stakeholders are only indirect users of the system or are affected only by the business outcomes that the system influences. Many are economic buyers or champions of the system. An understanding of who the stakeholders are, and their particular needs, is a key element in developing an effective solution.
Examples of stakeholders:

- Customer (or Customer representative)
- User (or User representative)
- Investor
- Shareholder
- Production manager
- Buyer
- Designer
- Tester
- Documentation writer, etc.

Figure 30: Description of Actor: Stakeholder according to RUP2000

In the recommendations, best practices and checklists connected to these workflows, the stakeholders and how to treat them and their needs is discussed and described on a very detailed level. Figure 31 below quotes the formal RUP definition of two actors on the customer side and the superclass of all the actors at the developer's side – the worker.

customer
A person or organisation, internal or external to the producing organisation, who takes financial responsibility for the system. In a large system this may not be the end user. The customer is the ultimate recipient of the developed product and its artefacts. See also stakeholder.

business worker
A business worker represents a role or set of roles in the business. A business worker interacts with other business workers and manipulates business entities while participating in business use-case realizations.

worker
A definition of the behaviour and responsibilities of an individual, or a set of individuals working together as a team, within the context of a software engineering organisation. The worker represents a role played by individuals on a project, and defines how they carry out work.

Figure 31: Extract from RUP2000 dictionary regarding actor terminology

The *business worker* is also regarded as an instance of the *worker*. In RUP 2000 there are listed 31 different workers, most of them at the developers side. Each of these workers has a specific role and responsibilities in the development process. One person can act as several workers. From the name of workers it is quite easy to roughly determine their high-level responsibilities. Figure 32 below lists the actors in the process. The actors at the developer's side in the figure that is marked with bold text are likely to be in direct contact with the customer actors.

Customer actors	Analysts	Developers	Testers	Managers	Additional workers
Customer -or Customer representative User - or User representative Investor Shareholder Production manager Buyer	**Business-Process Analyst** **Business designer** Business Model Reviewer Requirements Reviewer System Analyst **Use-Case Specifier** **User-Interface Designer**	Architect Architecture Reviewer Code Reviewer Database Designer Design Reviewer Designer Designer Integrator	Tester Test Designer	Change Control Manager Configuration Manager **Deployment Manager** Process Engineer **Project Manager** Project Reviewer	**Course Developer** Graphic Artist System Administrator Technical Writer Tool Specialist

Figure 32: Actors in RUP 2000

4.4.3 Messages in the Rational Unified Processess

Features that can be considered to be messages are the *reports* that are generated as momentary extracts from the artefacts. Many of these reports are used in order to describe and validate both the development process and the product under development. The purpose is to provide a base for the evolving agreement with the customer of what is to be developed. Thereby many of these reports must be regarded as messages for the customer-developer interface. Figure 33 lists the most important reports in RUP 2000. All the reports of the *business modelling set* and the *requirements set* is likely to be communicated to the customer and therefor highlighted in the figure.

Business Modelling Set	Requirements Set	Analysis & Design Set	Test Set
Business Use-Case Model Survey **Business Use-Case** **Business Object Model Survey** **Business Use-Case Model Realisation** **Business Worker** **Business Entity**	**Use-Case Model Survey** **Use Case** **Actor** **Use-Case Storyboard**	Design Package/Subsystem Design-Model Survey Use-Case Realisation Class	Test Survey

Figure 33: Reports in RUP 2000

The reports are not the only defined messages or message-related phenomena in the RUP. Figure 34 describes some of the other. These are defined in a way that makes them possible to treat in the configuration management system of the process.

change request (CR)
A general term for any request from a stakeholder to change an artefact or process. Documented in the Change Request is information on the origin and impact of the current problem, the proposed solution, and its cost. See also enhancement request, defect.

enhancement request
A type of stakeholder request that specifies a new feature or functionality of the system. See also change request.

stakeholder need
The business or operational problem (opportunity) that must be fulfilled in order to justify purchase or use.

stakeholder request
A request of any type – for example, Change Request, enhancement request, request for a requirement change, defect – from a stakeholder.

Figure 34: Extract from RUP2000 dictionary regarding messages

The low-level activity *Requirements Workshop* can be regarded as yet another aspect of messages. The main objectives of the Requirements Workshop is:

- To make the project team meet the stakeholders of the project.

- To gather a comprehensive "wish list" from stakeholders of the project.
- To prioritise the collected requirements based on stakeholders attending the workshop.

4.4.4 Analysis and Summary

To what extent is the RUP approach to address the customer-developer interface possible to capture in the three theoretical perspectives? The *process perspective* and *the actor perspective* are very easy to use when describing the customer-developer interface according to RUP. The RUP model is indeed fully aware of the communicative benefits of an iterative development process. A lot of interaction points towards the customer are defined. The main actors are described and the model even offers low-level instructions in how to find and analyse the actors, their roles and responsibilities.

A lot of messages were also possible to find. At a low level of the RUP there are also a lot of *what, when* and *how* regarding messages in the customer-developer interface.

The mapping between the process and the three theoretical perspectives seems to be good[10].

One aspect of RUP that partly can be related to messages is the recommendation of using the Unified Modelling Language (UML) when modelling, designing and communicating the system. If the advocates of UML are right, the using of UML as a common language for messages between the customer and the developer during the development project, will mitigate problems corresponding to deficiencies in customer knowledge in the developer's domain and the developer's knowledge of the customer's business domain. The use of UML as a strategy for reducing the conceptual gap between application domain and the developer's engineering domain was not captured by the theoretical perspectives.

4.5 Extreme Programming – XP

Extreme Programming is a young development process model that has gained a lot of attention in the software development society. The first chapter of the originator Kent Beck's book "Extreme Programming explained" is introduced by the statement:

> "Software fails to deliver, and fails to deliver value. This failure has huge economic and human impact. We need to find a new way to develop software" [Beck 1999, p 3]

That is what Extreme Programming (XP) claims to have done.

[10] For a process this complete and all embracing it is worth noticing that in the *support workflow: environment* there is an *actor: process engineer* with the responsibility to design and develop the specific process for the software development project. It is interesting to notice that in none-of those workflow details the stakeholders are mentioned. RUP does not seem to have a plan for how to design the customer-developer interface – perhaps a challenge for the future.

The stereotyped picture of the XP process says that if everything is:

- tested every day,
- the system is expressed in pure code
- and there is a full time customer in the development room,

then will the developed software be delivered:

- with desired functionality,
- on scheduled time,
- within budget,
- to an excellent quality.

XP is more sophisticated than that. Figure 35 gives a view of key features of the XP process and how they support each other.

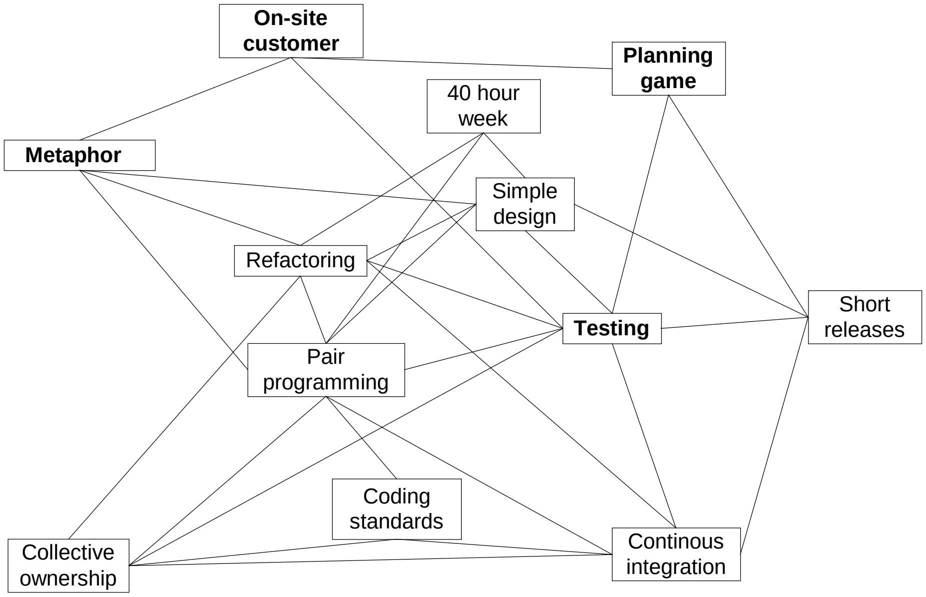


Figure 35: Key features of Extreme Programming and their supportive interconnections[11].(Beck 1999, p 70)

Those features that have direct connections to the customer-developer interface are highlighted in the figure and will be discussed here.

- *On site-customer* gives the development team continuos access to the customer competence and possibility to let the customer make fast decisions and priorities.
- The *planning game* is a continuously running planning activity where the customer prioritises among features for implementation.
- Continuous customer-driven *testing* gives the customer confidence in the software under development.

[11] "*Refactoring* – Programmers restructure the system without changing its behaviour to remove duplication, improve communication, simplify or add flexibility." (Beck 1999, p 54)

Collective ownership address collectively within the group of programmers.

- The term *metaphor* stands for the need of common language that both the developers and the customer has to use in order to understand each other.

Two other features related to the customer-developer interface are:

- The customer writes *stories*, which can be described as the XP-variant of the *use-cases*.
- Splitting of the power and responsibility between the customer and developer. The customer decides on timing of releases, makes priorities among features (the stories) and decides the exact scope of each feature. The developer has to provide information about technical and schedule consequences of various features and choices.

4.5.1 Process Model of the Extreme Programming

A typical development process of an XP-project can be divided into different phases [Beck 1999, chapter 21]. These phases can be regarded as high-level activities. The phases and their contents are listed in XP terminology below:

- **Exploration** – Gaining confidence in the possibility to reach the project objectives, trying out tools and technologies, experimenting with architectural ideas etc. The customer practising writing *stories.*
- **Planning –** Planning the first development cycle, the customer prioritises among the stories, sets date for the first release.
- **Iterations** – The customer writes functional tests together with the Tester. The programmers design, implement, and integrate the software based on the customer written stories chosen in the Planning Game. Perform a planning game for the next iteration.
- **Productionising** – Deployment of the first release in the target environment.
- **Maintenance** – Keep the system running in the target environment and at the same time implementing new features and continuously test and refactor the system.
- **Death** – Project termination. Document the system.

Figure 36 below illustrates how XP high-level activities relate to each other and to the customer-developer interface. In order to illustrate that the customer is a part of the project team, all activities are gathered in one box.

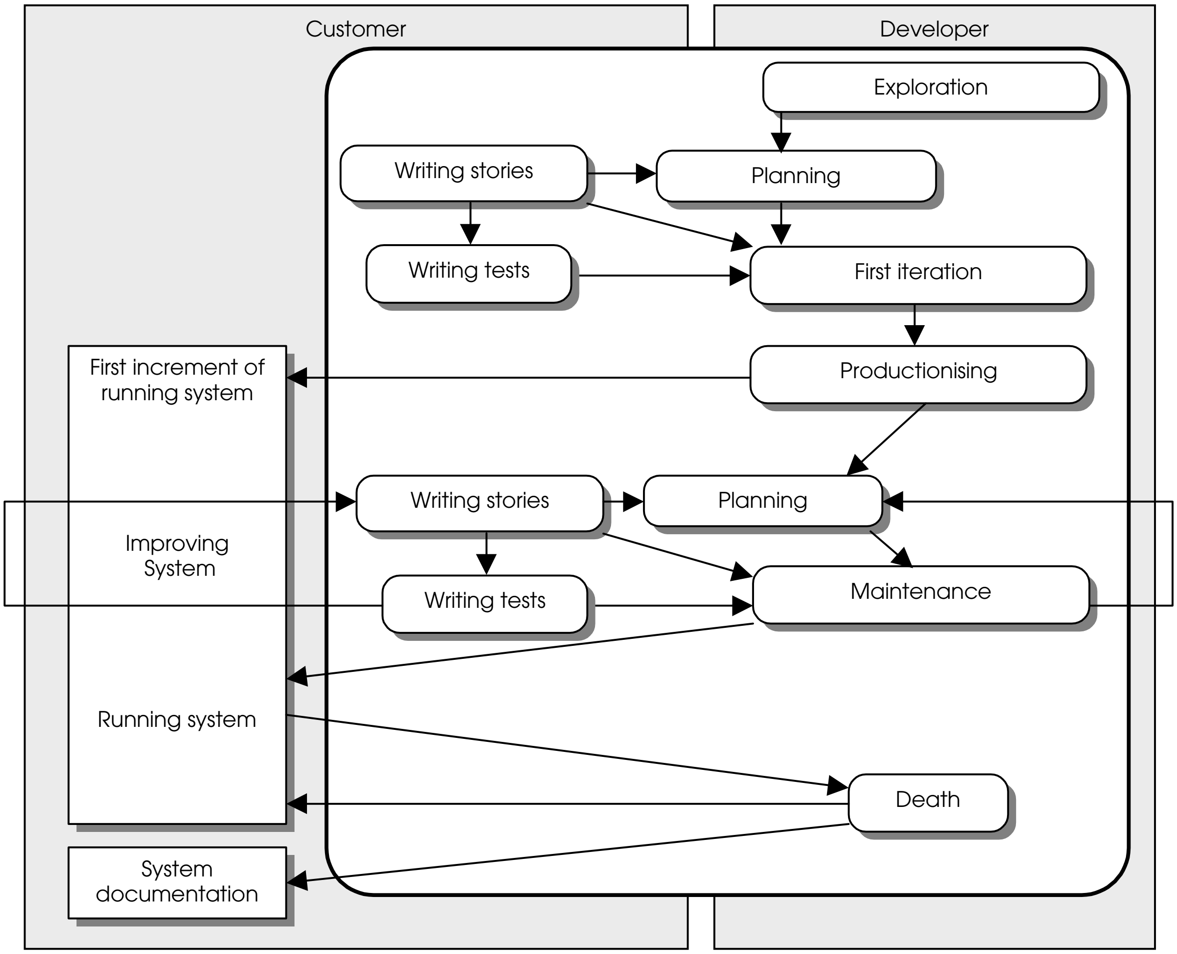


Figure 36: Process model of a possible XP-project

4.5.2 Actors and Stakeholders in Extreme Programming

The description of XP [Beck 1999, chapter 22] includes a listing of the actors in an XP project.

Actor	Roll
Customer	Provides domain knowledge during the project by writing stories and makes decisions and priorities among features. Designs functional tests.
Programmer	Designs. Implements. Integrates. Tests. Refactors the system. Informs the customer on technical aspects of the stories. Teaches the customer to write stories to ground priorities.
Tester	Helps the customer in writing tests.
Tracker	Monitors, evaluates and evolves the development process .
Coach	Guides and inspire the development team.
Consultant	Technical expert on matters that the development team does not master.
Big Boss	Monitors and defends the project. Terminates the project if it fails.

Figure 37: Actors and stakeholders in Extreme Programming

4.5.3 Messages in Extreme Programming

If it is easy to picture the high-level process model and to identify the actors of XP it is harder to identify specific messages. One of the most central concepts of XP is the continuos oral communication between the on-site customer and the programmers regarding visions, requirements, clarifications, technical complications etc. The stories describing the scope of the system are messages from the customer to the developer. Preferable everything else within the process is expressed in code or in executable software. The test cases that the customer designs can be regarded as messages in a validation process. Figure 38 below lists the main messages identified in XP.

Product information	**Source**	**Receiver**	**When**	**Media**
Continuos project communication	Customer Developer			Oral
Stories	Customer	Developer		Structured textual document
Estimations on time and complexity etc	Developer	Customer		
Decisions, priorities etc	Customer	Developer		
Tests	Customer	Developer		Code
Running System	Developer	Customer		Executable software
System documentation	Developer	Customer		
Process information:				
Development plan	Customer Developer			Written document

Figure 38: Messages in XP

4.5.4 Analysis and Summary

XP could comfortably be captured by the process model perspective and the actor and stakeholder perspective. But the way to find out the main messages communicated between the customer and developer was not straightforward.

XP did not introduce any new high-level activities. But it did introduce some new features of the high-level activities and of the roles of the stakeholders that affects the customer-developer interface. Both of the new features deal with strategies for reducing conceptual gaps.

First, the concept of letting the customer design tests reduces one of the gaps described in the Gaps-model (see Section 4.3) – the difference between what the *customer-perceived need* and *the customer-perceived result*. By letting the customer design and own the functional tests, the customer can be said to continuously monitor and calibrate the gap between the *need* and the *result*. Either by modifying the *need* towards what *may be implemented* or by affecting what is *implemented* in the direction of *the need*.

Second, the concept of bringing the customer into the development team is an attempt to reduce conceptual gaps between the customer and the developer side regarding the application domain and the software development domain. A continuos fraternisation and the short communication links help both parties to better understand each other's expertise.

But the on-site customer also introduces a new problem. XP says that it is important that the customer is capable of representing the end-user and is empowered by the customer organisation to make fast decisions throughout the project. If the stakeholders or the goals of the project are in conflict or if the application domain is very large, complex and dynamic, it is very hard to find such a person. Add to that the fact that living with the programmers for a long period of time probably will push the loyalty of the customer representative in the direction of the developer organisation.

Hence there is a risk that the customer representative becomes a kind of hostage for the developer organisation, successively moving the customer-developer interface to a point between the customer representative and the organisation of the customer representative. In the short run this could be good for the developer that can claim that the customer is satisfied, but sad for the customer organisation that might get the wrong system.

The *metaphor* concept could to some extent be mentioned in the same context as the conceptual gap and the RUP UML since the basic idea of the metaphor is to establish a common language between the customer and the development team. The theoretical perspectives did not capture this aspect.

4.6 Dynamic System Development Method – DSDM

The origin of Dynamic System Development Method (DSDM) is a consortium founded and managed by large software development companies mainly focused on *in-house* development of *business applications*. The in-house focus means that the terms for the parties in the process model often are the *user* and the *IT-department*. In this report these terms often are translated to *customer* and *developer* respectively, in order to facilitate the reading and analysis. The information about DSDM in this section is based on the book "DSDM Dynamic Systems Development Method", by Jennifer Stapleton [Stapleton 1997]. In the book she places a warning for using DSDM in contractual development:

> "Many external suppliers are using DSDM: it can work but there has to be trust on both sides…" [Stapleton 1997, p17].

Despite this warning DSDM has features that are suitable for the analysis of the customer-developer interface.
Just as in the case with RUP, the details of DSDM are only available to paying members. The book [Stapleton 1997] often refers to a *manual.* The contents of that manual is unfortunately unavailable for this study.

The nine underlying principles of DSDM are listed in Figure 39 below.

Underlying principles of DSDM

1. Active user involvement is imperative
2. DSDM teams must be empowered to make decisions
3. The focus is on frequent delivery of products
4. Fitness for business purposes is the essential criterion for acceptance of deliverables
5. Iterative and incremental development is necessary to converge on a accurate business solution
6. All changes during development are reversible
7. Requirements are baselined at a high level
8. Testing is integrated throughout the lifecycle
9. A collaborative and co-operative approach between all stakeholders is essential

Figure 39: Nine underlying principles of DSDM (Stapleton 1997, p 11-18)

The basic ideas of DSDM have a lot in common with the ideas of XP. Incremental flexible development, closeness to the customer and short release cycles should be combined with continuos testing and an early first release covering those features that has the highest value to the business of the customer. The project team, which includes a customer representative, has to be empowered to make decisions. Compared to XP, DSDM shows a higher dependency of, non-code documentation and formal configuration management. It is for instance very important that all changes to the software are reversible. DSDM also gives instructions and recommendations about when to do what on a lower level than XP does. To that extent DSDM has more in common with RUP than with XP.

4.6.1 Process Model of the DSDM

The process model of DSDM consists of five high-level phases briefly listed and described below:

1. **Feasibility study –** The objective of the feasibility study is to, on a high level find and explore the objectives of the project and to evaluate whether it is technically possible to implement the system or not. Another purpose is to find out if DSDM is the right methodology for the project.

2. **The business study –** The objective of the business study is to picture the environment of the system and create the Business Area Definition document including high-level object models, data flow diagrams and entity relation diagrams.

3. **Functional model iteration –** In this phase the focus is set on refining the business aspect of the system. The phase is iterative, both design documents and software is created and evolved to a point where they, by reviews, prototypes or tests, are determined to have reached the goals for the phase.

4. **Design and build iteration –** In this phase the system is refined and tested to the quality necessary for deployment.

5. **Implementation** – During the implementation phase the system is deployed in the target environment. System and user documentation is delivered and the end-users are trained to use the system.

Phase three, four and five are iterative. It is also possible to move between the phases e.g. to jump from phase five to phase three or to phase four.

Users outside the development team are engaged in requirements workshops and when the development team needs special opinions. The main contribution of application domain competence is accomplished by that the:

> "… process involves a few knowledgeable users who support or participate in the development team throughout the project. This is as opposed to the traditional approach of sending out documents to a mass of users and calling in a fairly large user population for acceptance testing at the end of the process." [Stapleton 1997, p 11]

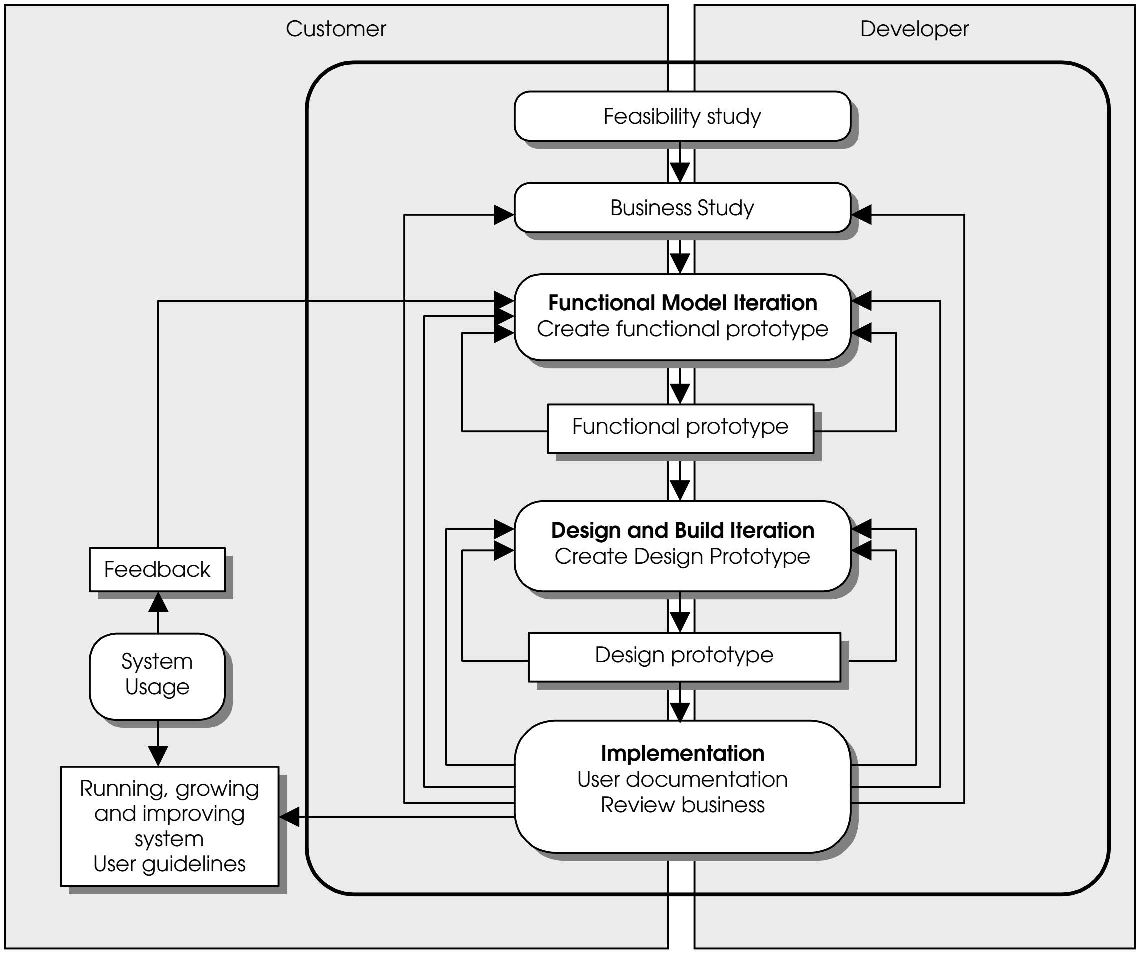


Figure 40: Process model of the customer-developer interface according to DSDM

The relative frequent introduction of new increments to the user society guarantees just as frequent validation of the implemented features. Figure 40 above illustrates how top level activities of DSDM relate to each other and to the customer-developer interface. In order to illustrate that the customer is a part of the project team all of the team activities are gathered in one box.

4.6.2 Actors and Stakeholders of DSDM

In the DSDM book [Stapleton 1997, Section 5.3] the main actors and stakeholders are described. They are listed in Figure 41 below.

Actor	Role
Ambassador User	Full time in the development team. Brings the knowledge from the user community into the team. Informs the other end-users of what is happening inside the team
Visionary	Participates in meetings to make sure that the team do not lose sight of the originate business objectives
Advisor User	Covers the ambassador user with disparate views of the system. On an ad hoc basis.
Executive sponsor	The ultimate decision maker and purse holder
Senior developers	Team or competence leader
Developers	Designs. Implements. Tests….
Technical co-ordinator	Architectural responsibility. Ensures technical consistency and the quality of the configuration management

Figure 41: Actors and stakeholders in DSDM

4.6.3 Messages in DSDM

In the section of *better communication* the author [Stapleton 1997, p 51] argues that a way to avoid mental barriers between users and developers is to reduce the amount of documents ridden by IT-jargon in favour of direct oral communication. Another concept recommended by DSDM is joint application design workshops, where end-users and the development team work together in designing the system. These workshops can be regarded as complementary when the capability of the Ambassador User and the Advisor User is not sufficient to bridge the mental gaps. A third message related feature introduced by DSDM is the concept of training the actors in their role in the development process so all the parties better understand what to do, when, how and why.

The fourth highlighted messages are the prototypes, used for validating that the developer develops what business needs. Figure 42 lists these messages.

Product information	Source	To	When	Media
Continuos project communication	Customer Developer		Always	Oral
Prototypes	Developer	Customer	---	---
Design workshops	Customer Developer		On demand	Workshop
Process information:				
Development process training	Customer Developer		Initially	---

Figure 42: Some messages in DSDM

4.6.4 Analysis and Summary

Could the three theoretical perspectives capture the customer-developer interface of DSDM? Since the model primarily is designed for in-house development there is, strictly speaking, no customer-developer interface. Besides that restriction, all of three theoretical perspectives could reveal relevant information about the customer-developer interface as seen by DSDM.

The risks of putting the customer in the development team that was identified when analysing XP (see Section 4.5.4 at page 70) are also present in DSDM. Perhaps the using of formalised *requirements workshops* has the ability to mitigate the risks.

A new objective was identified: To mitigate problems caused partly by wrong or unrealistic expectations of the development process and partly by participants not understanding their role in the project. The concept of training the actors in how they expect to behave in the development process was a practice introduced to reach the new objective.

4.7 Summary and Evaluation of the Customer-Developer Interface According to the Best Practices

The objectives of this section (4) were the following:

1. extract objectives and practices for the customer-developer interface from established best practices of software development and
2. evaluate the usefulness of the three theoretical perspectives from Section 3.

All six best practices describe the software development process with different objectives, with unique terminology and from different perspectives. Their differences make it hard to compare the pictures of the customer-developer interface they give. In order to make it possible to both overview and compare the objectives and the practices from the best practices, the terminology has to be streamlined. Figure 43 below is a summary of the objectives and practices of the six examined best practices, sorted by *objective, best practice* and *practice* in descending order.

Interface objective	Best practice	Practice	Captured by
Customer confidence in the development process	SA CMM	Communicate process status to customer Audits. Process reviews.	Process
	RUP	Workflow: deployment	Process
Customer confidence in the product	SA-CMM	Customer driven acceptance test	Process
	Gaps-model	System explanation	Message
	XP	Customer driven testing	Process
Customer validation of developer specification	SA CMM	Evaluation of product	Process
	RUP	UML	---
	DSDM	Prototypes	Message
Customer understanding of the development domain	SEI TRM	Joint Organisation	Actor
	SEI TRM	Risk Management	Process
	RUP	Reports	Message
	RUP	UML	---
	XP	Metaphor	---
	DSDM	Requirements workshops	Message
Actor understanding of roles	DSDM	Process training	Message
Flexible project adjustment to evolving requirements	SA CMM	Evaluation of product	Process
	RUP	Incremental development	Process
	XP	Incremental development	Process
	DSDM	Incremental development	Process
Flexibility of customer developer interface	SA CMM	Customer driven process modification	Process
	SEI TRM	Joint Risk Management	Process
Developer understanding of customer business domain, needs expectations and requirements	SEI TRM	Joint organisation	Actor
	Gaps-model	Marketing surveys	Process
	RUP	Workflow: business modelling	Process
	RUP	Workflow: requirements	Process
	XP	On-site Customer	Actor
	XP	Metaphor	---
	DSDM	The business study	Process
	DSDM	Design workshops	Message
	DSDM	User Ambassador	Actor
Team spirit	SEI TRM	Joint organisation	Process
	XP	Continuos Communication	Message
	DSDM	Continuos Communication	Message
Common product vision	SEI TRM	Joint organisation	---
	XP	The metaphor	Message
Close customer expectation-perception-gap	Gaps-model	Entire model	---

Figure 43: Summary of objectives and practices from the examined best practices

The left column of Figure 43 represents syntheses of qualitatively different objectives for the customer-developer interface that was found in the six best practices. These objectives partially overlap one another in terms of concept. The second column represents best practices that have a practice for achieving the objective. The third column represents a selection of practices for achieving the objective, formulated in the terminology of the best practice. One and the same practice also appears as methods for achieving several of the objectives. Together the *objectives* and the *practices* form a comprehensive picture of objectives of a proper customer-developer interface and how it could be accomplished.

The second objective of the section (4) was to evaluate the usefulness of the three theoretical perspectives. One way to do that is to evaluate hits in Figure 43 above. The right column shows which perspective that can be classified as primarily responsible for capturing the practice. Since there is no indication of which of the identified objectives for the interface is most important, there is no point in counting the number of hits. It is enough to verify that all three perspectives are represented in the right column and state that the perspectives are useful when describing, analysing and comparing customer-developer interfaces.

The empty spaces in the column are more problematic. They represent those objectives and practices that were identified out of sight from the theoretical perspectives. A feature that was not directly identified by the perspectives was *the conceptual gap* described in the Gaps-model and partly adressed by features as *RUP: UML* and perhaps also by *XP: metaphor*.

There is no secure way of knowing if other relevant features were missed. But the three theoretical perspectives helped to capture enough aspects of the customer-developer interface to motivate the use of them in assessment and analysis of real life projects.

A strategic methodological decision to choose exactly six best practices, not eight or three was made early in this section (page 44). The objective “Actors understanding their role” was identified in only one of the best practices (DSDM). That is also valid for the objective “Reduce the conceptual gap between customer’s expectations and perceptions”. The latter objective was found in the Gaps-model. All the other objectives could be identified in more than one best practice. There is no reason to believe that some of the objectives identified in the best practices should be regarded as more relevant and important than others, but perhaps the nine objectives identified in more than one best practice could be regarded as somehow stronger than the other two objectives?

The Gaps-model is not primarily focused on software development. This protruding best practice revealed an objective (the conceptual gap between customer expectations and perceptions) that was not obvious in the other five best practices. This finding contradicts the idea (presented on page 44) that the trade-off between the chance of finding new features and the cost of investigating further best practices landed on the six chosen best practises. The five best practices chosen from the domain of software development where probable many enough; but investigations of best practices outside the domain of software development could give further input to the list of objectives and practices. On the other hand, to widen the circles too far from the discipline of software engineering could obstruct the arguments for the strategic decision for limiting the scientific perspectives (see Section 1.4.5). Perhaps adding additional best practices would make the result harder to use for the thought of target group (see Section 1.2)?

5 Customer-Developer Interfaces in Real Life Projects

Recall from Section 2.3 at page 28 that there was a double objective of investigating real life customer-developer interfaces:

1. to test the usability of the theoretical perspectives of Section 3 when assessing and analysing real world customer-develop interfaces
2. to cross-verify that the problems and practices of the customer-developer interface extracted from established best practices of software development has correspondence in the real world projects.

In order to reach these objectives six customer-developer interfaces of real life projects has been assessed and analysed based on the three theoretical perspectives. The customer-developer interfaces are described in Appendix A. Before presenting the results of the assessment and the analysis of the projects, the foundation for the choice of projects and the method for assessment is presented.

5.1 Choice of Projects

At a first glance it would seem suitable to investigate projects with characteristics from upper left corner of Figure 3: Customer and developer maturity matrix, since those are the target projects of the study. Probable findings would be a lot of interface-related problems and better or worse ad-hoc actions in order to mitigate the problems. But the purpose of the study is not to verify if projects with ad-hoc customer have problematic interfaces. The purpose is to find strategies to mitigate problems of the customer-developer interface as listed in Figure 43. The next thought is therefor to investigate successful projects from which it would be possible to learn some usable practices. This second approach would on the other hand bring about the risk of finding practices that are hard to apply on projects with ad-hoc customers. The proper consequence of this reasoning is to both study inexperienced ad-hoc customers and projects with experienced customers.

In order not to get stuck in corporate culture, terminology etc. the six projects were chosen from two different development companies. Senior management persons in each of the two companies with interest in and understanding of the possible complications in customer relations, chose the projects. Their main objectives of investing corporate time and energy in the study was to get feedback on how their businesses behave in the customer-developer interface and to get methodological inspiration from the result of the study.

5.2 Assessment Methods

The projects were assessed in the form of structured interviews. Interviewees were the project managers at the developer organisation. Dates and interviewees for the interviews are listed in Section 8.1 at page 89. Each interview lasted for approximately one hour. The main areas for the interviews are directly reflected in the structure of the project descriptions as in appendix A:

- Project scope and overall organisation
- Process model of the customer-interface
- Actors-and stakeholders on the customer and developer side respectively
- Messages communicated between the parties
- Experienced problems and chosen solutions to the problems.

In project A and project B the interviews were combined with written documentation in the form of project specifications and project logs etc.

The interviews was documented in the same form and syntax as in Appendix A and sent for validation to the interviewed project manager. The customers are made anonymous. The terminology in the documentation is mostly based on the terminology used by the interviewee. After a couple of days the document was validated by a telephone walkthrough with the interviewee. After changes the descriptions and analysis of the interfaces was put in baseline as in Appendix A.

5.3 The Projects

The objectives of this section were the following:

1. Answer the question: Could the three theoretical perspectives of Section 3 capture the essence of the customer-developer interfaces in a way that make it possible to compare them to each other and extract essential experiences from the projects?

2. If so, compile the experiences from the customer-developer interface in a way that is possible to overview.

Answer to the first question: Yes, it was possible to capture essential aspects and experiences of real life projects by using the three theoretical perspectives. The argument for this is quality of the descriptions and analyses of the real life projects, as shown in Appendix A.

Many features and characteristics of the interfaces were similar or almost identical between many of the projects. Since there were only six projects in the study, it is impossible to draw any generic conclusions; however these six projects have the following features in common:

- Initial requirements are not detailed enough to develop a system from. The customer requirements evolve throughout the project.
- The projects are iterative.
- It is hard to force the customer to communicate requirements, change requests and problem reports in a structured form.

Project	Application domain	Problems	Characteristic or new features
A	Recruitment agency	• The initial Requirements Workshop was badly communicated to the project. • Customer business processes developed in parallel with the system. • Organisational problems at the customer site • Unreliable customer representative • Limited end-user involvement	• Project Communication Plan • Attempt to formalise problem reports and change requests
B	Customer internal service broker	• Waterfall model could not be followed • Written specifications etc could not be validated • Double communication from co-contractor	• Specified increments • External Head Project Manager as co-ordination • Direct contact between developers and expert-users • Live walkthroughs of specification and prototypes • Large acquisition organisation at the customer side
C	Web-shop for system components	• Initial problems regarding structure of requirements	• Planned incremental development • Customer educable regarding structure of requirements • Formal specifications could be validated by the technically skilled customer • Formal system reports • End-user feedback via help-desk and user training
D	Web-shop for consumer product	• Undefined co-contractor relations	• Empowered development team • Limited financial constraints • Team spirit
E	Internet Encyclopaed ia	• Parallel and dependent software development project	• Incremental development • Many co-contractors • Many technical interfaces • Usability consultant
F	Internet magazine	• Evolving requirements • Payment delays	• Pilot study • Process training • Prototype walkthrough

Figure 44: Project scope, problems and characteristics

In order to get an overview of the six projects and their application domain; a selection of problems and characteristics is listed in Figure 44 above. The selection of *problems* and *characteristics* from the material in Appendix A is based on the intention to extract qualitatively unique aspects. This means that problems or characteristics listed on one project may also have been present in another project. Some of the characteristics and features of the right column are solutions to the problems in the problem-column. However it is not always possible to identify an exact match between a problem and a solution - they just appear in the same projects.

5.4 Real World Projects vs. Best Practices

The purpose of this section is to compare the findings in the *best practices* with the findings in the *real life projects.* At a high level it is easy to conclude that the collected thinking of the best practices is reflected in the experience of the six assessed projects. But the match is not perfect. Some problems identified in the real life projects were addressed in the best practices in a rather straightforward way. Figure 45 matches problems from Figure 44 with suggestions for practices from Figure 43 (page 76) that address the same problem.

Problems	Practice	Best practices
The initial Requirements Workshop was badly communicated to the project.	Artefact reports	RUP
Customer business processes developed in parallel with the system.	Incremental development	RUP, XP, DSDM
Organisational problems on the customer side	---	
Unreliable customer representative	---	
Limited end-user involvement	Workflow: business modelling Workflow: requirements	RUP
	Marketing surveys	Gaps-model
Waterfall model could not be followed	Incremental development	RUP, XP, DSDM
Written specifications etc could not be validated	Prototyping	DSDM
Double communication via co-contractor	---	
Initial problems regarding structure of requirements	Process training	DSDM
	UML	RUP
Undefined co-contractor relations	---	
Parallel and dependent software development project	---	
Evolving requirements	Incremental development	RUP, XP, DSDM
Payment delays	---	

Figure 45: Problems in real life projects vs. practices from the best practices

The practices in Figure 43 address more problems than those identified in the six projects. This does not mean that there is no need for those practices, just that the problems they were intended to solve or mitigate were not identified as problems in the six studied projects. If the study had covered more projects, perhaps more of the practices had come to use.

A complication is that some of the problems identified in the real life projects were not dealt with in the best practices. The empty spaces in Figure 45 represent real life problems that have *no clear match* among the objectives and practices listed in Figure 43 (page 76). However, just because no exact match was found does not mean projects cannot learn from using best practices. A quick and easy suggestion for solving the equation is that a proper use of SEI Team Risk Management could have identified the remaining problems, allowing the projects to take actions to avoid them. The organisational problems at the customer site and the undefined co-contractor relations could perhaps have been identified by applying RUP Business Modelling or DSDM Business Analysis of the project actors and stakeholder themselves, instead of just of the application domain. DSDM Process Training could perhaps force the ad-hoc customer to play an active part in monitoring and controlling of the project and thereby enforce a better definition of the relations between co-contractors.

Another reaction to the fact that some of the identified problems could not be matched to a practice is to re-examine the six best practices of Section 4. If that is not enough the search for new best practices could be re-started.

Assessment and analysis of the real world interfaces revealed new candidates for the list of *objectives* for a proper customer-developer interface (Figure 43). One of them, synthesising the three problems regarding co-contractor relations could be: *Developer understanding the project environment.* The other new objective, concerning the trustworthiness of the customer could be something like: *Developer confidence in the customer*. It is possible to find ideas for solutions to these problems in the six real world customer-developer interface of in Appendix A. But as indicated in the beginning of Section 5.3, those practices are not validated in the same way as the ones from the six best practices; hence it is better to let the new objectives go unanswered.

A third reaction to the identification of new interface objectives is to highlight the three theoretical perspectives. It was the use of the perspectives when analysing real world cases that revealed new objectives for the customer-developer interface. This further confirms the usability of the three theoretical perspectives when capturing, describing and analysing customer-developer interfaces.

5.5 Summary

In order to verify the findings in the established best practices this section has showed how six real world customer-developer interfaces has been assessed, described (Appendix A – Cases Of Customer-Developer Interface) and analysed in the light of the three theoretical perspectives. The findings in the real world cases were then compared to the findings in the best practices.

Many of the problems of the real life projects were addressed in some of the best practices. Some problems in the real life projects had no straightforward solution in the best practices but well some indirect solutions by e.g. Team Risk Management. Two new objectives for the customer-developer interface were extracted from real world projects:

- Developer understanding the project environment.
- Developer confidence in the customer

The usability of the three theoretical perspectives was further verified.

6 Summary and Conclusions

The objective of this section is to summarise the study and its findings and to draw conclusions by evaluating the hypothesis from Section 2.3 (page 27) and answer the question:

> Can use of the theoretical perspectives, together with lessons from the best practices and real world customer-developer interfaces guide us to a better understanding of how to plan the interface to ad-hoc customers?

6.1 Summary

The objective of the study was to explore the customer-developer interface in development projects with ad-hoc customers and to summarise best practices. This has been achieved in the following steps.

1. Three theoretical perspectives for how to capture and describe a customer-developer interface were outlined and described:
    - Process Model Perspective
    - The Actors and Stakeholders Perspective
    - Message Perspective

2. The objectives and practices of customer-developer interface were extracted from the six *best practices* of software development:
    - SEI Software Acquisition Capability Maturity Model – SA CMM
    - SEI Team Risk Management
    - The Gaps-model
    - Rational Unified Process - RUP
    - Extreme Programming - XP
    - Dynamic Systems Development Method – DSDM

    The main objectives for a well functioning customer-developer interface according to the six best practices were found to be:
    - Customer confidence in the development process
    - Customer confidence in the product
    - Customer validation of developer specification
    - Customer understanding of the technical domain
    - Actor understanding of roles
    - Flexible project adjustment to evolving requirements
    - Flexibility of customer developer interface
    - Developer understanding of customer business domain
    - Team spirit
    - Common product vision

- Close the customer's expectation-perception-gap

Figure 43 on page 76 gives a "smorgasbord" of practices from the six best practices that address the objectives. The three theoretical perspectives were found to fulfil their purpose. However, matters concerning conceptual gaps between the customer's expectations and perception and between the customer's business domain and the developers technical domain could not easily be captured by the theoretical perspectives.

3. In order to validate the findings in the best practices, six real life projects where assessed, described and analysed in terms of three theoretical perspectives. The findings in the real life projects were then compared to findings in the best practices. Many of the problems of the real life projects were addressed in some of the best practices. Some problems in the real life projects had no straightforward solution in the best practices but some indirect solutions by e.g. Team Risk Management. Two new objectives for the customer-developer interface were extracted from real world projects:
   - Developer understanding of the project environment.
   - Developer confidence in the customer

   No validated practices for achieving the new objectives were identified.

6.2 Evaluation of the Objective of the Study

One objective of the study was to explore the customer-developer interface in software projects with ad-hoc customers. This objective has been achieved by first describing three theoretical perspectives and then using the perspectives to assess, describe and analyse real world projects.

The other objective was to summarise best practices of customer-developer interface. This objective has been achieved by extracting interface-related objectives and practices from six different best practices of software development.

6.3 Conclusion

The hypothesis from Section 2.3, page 27 was formulated:

> If a customer-developer interface is described by its process model, its actors and stakeholders and its messages, this description is all-embracing enough to give a relevant understanding of how to plan and evaluate a customer-developer interface.

The three theoretical perspectives were very useful when capturing, describing and analysing the customer-developer interface even if they missed some conceptual aspects. If what the three perspectives identify can be considered as *good enough* is a matter of judgement that determines to which degree the outcome is relevant. If it is *good enough*, it is clear that the use of the theoretical perspectives, together with lessons from the best practices and real world customer-developer interfaces, guides us to a better understanding of how to plan and evaluate the interface to ad-hoc customers.

Figure 43 at page 76 lists the objectives for the planning of a customer-developer interface. The same list is also a quick index for where to find strategies or practices for achieving those objectives.

When waiting for further research (see Section 7.2 page 86) the two objectives found in the real world projects could be added to the list of objectives from the best practices. If the word “not” was put in front of each objective, the list would be an excellent initial list of risks in a risk management activity, hopefully in tandem with the ad-hoc customer.

Another recommendation to the account manager or project manager who plans a development project with an ad-hoc customer is to identify and analyse the actors on the customer side. The list of possible actors in Section 3.2 seems to be complete. Ideas for how to find the actors are given by *Workflow: business modelling* in RUP. If possible, have an on-site customer, but be aware of the risk of creating a new interface between the on-site customer and the customer’s organisation.

The next recommendation is to understand that the needs, expectations and requirements *will* change. Plan for iterative development. Make a plan together with the customer for most of the messages identified in Appendix A.

We have been guided to a better understanding of how to plan an interface to an ad-hoc customer. There is no explicit reason to be surprised by something that can be clearly foreseen.

7 Further Research

Answers often raise new questions. This section presents some of these questions and argues for why it would be interesting to have them answered.

7.1 Model the Conceptual Content of the Customer-Developer Interface

How can knowledge, ideas and visions be described in a form that is easy enough for the customer to formulate and structured enough for the developer's needs. The Unified Modelling Language helps the developer to present his interpretation of the business domain in a form that the customer is able to validate. But UML is hardly the syntax the non-technical customer may use for presenting an initial vision for the system. The idea of the XP Metaphor seems to be more duplex-oriented, but a very rich common language that works within a development team may have limitations when used for communicating with a complex business domain.

One practical and probably often used solution to this problem is to engage people with double expertise, who will personify the conceptual interface. These are the XP on-site customers and DSDM Ambassador Users on the developer side. They might, on the customer's side, be a technically skilled project manager as in Project B or a SA-CMM level 5 acquirer. But as indicated e.g. in the books about XP [Beck 1999] and DSDM [Stapleton 1997], those people are short in supply. For very complex system, demanding expertise on several domains, their mission is impossible.

If it was possible to develop this unified *language for ordinary people*, formal communication regarding needs, requirements, expectations, specifications and change requests would be easier to develop. Such a system could be a part of the developers configuration management system. It would solve problems that were indicated in many of the real world projects of this study.

7.2 Find Practices for the Unaddressed Objectives

The comparison of the findings in the best practices and the findings in the six real world cases ended in the discovery of two objectives for the customer-developer interface that was not addressed by any practice identified in the best practices:

- Developer understanding of the project environment.
- Developer confidence in the customer

A suggestion for further research is to find practices for achieving them by reinvestigating the six best practices and widen the circle of best practices.

7.3 Evaluate how to Quantify of the Objectives of the Interface

Every ad-hoc acquisition is by definition unique and it is possible to question the usefulness of quantitative assessment. See Section 1.2.1 at page 15. At the same time one has to understand that the status of being an ad-hoc customer is relative. Many of the six leal world projects described in Appendix A run on rather long terms. What was intended as a fast one-shot system became a relationship of many years' duration. For this reason it would be very interesting to evaluate what aspects of the interface objectives in Figure 43 and the two new objectives identified in the real world projects, that is possible to quantify. If it were possible to set up quantitative measurable objectives for the interface, it would as a consequence also be possible to monitor and evolve the joint performance of the parties.

A second possibility that would emerge from quantifying the objectives of the customer-developer interface could be to execute the quantitative analyses outlined in Section 2.1 at page 23.

7.4 Structure the Messages of the Customer-Developer Interface

As a means for analysing the customer-developer interface many different types of messages has been identified. As they are listed in the report they may serve as inspiration or unstructured checklists of messages that are more or less likely to be communicated during a development project. It would be very interesting to see a more structured and compiled list of better defined messages. The list would probably be an excellent base for the communication plan of a development project, dealing with simple things, as which persons send and gets what messages. The strategy to achieve the list is the same as used in object modelling: identify, compare, sort, structure, find names for classes and categories and make sure that both senders and receivers have the same conception of which message is what.

8 References

[ArianeV96] http://www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html, Date: Jan 28, 2001

[LAS 1995] http://www.cs.ucl.ac.uk/staff/A.Finkelstein/las.html, Date: June 10, 2001

[Beck1999] K. Beck, “extreme Programming explained EMBRACE CHANGE”, Addison-Wesley, 2000

[Gibbs 1994] W. Wayt Gibbs, ”Software's Chronic Chrisis”, Scientific American, September 1994, pp. 86-95

[CMMI 1.02d] CMMI ^{SM} for Systems Engineering/Software Engineering/Integrated Product and Process Development/Acquisition, Version 1.02d DRAFT CMMI ^{SM} -SE/SW/IPPD/A, V1.02d DRAFT

[CMU/SEI-94-SR-5] P. R. Higuera, A. J. Dorofee, J. A. Walker, R. C. Williams “Team Risk Management: A New Model for Customer-Supplier Relationships”, Special Report CMU/SEI-94-SR-5, July 1994

[CS 1999:10] K. Karlsén “Missuppfattning äventyrar pensionssystemet”, Computer Sweden 1999:10

[CS 1999:49] K-J. Bytner “Pensionssystemet ses som högrisk project”, Computer Sweden 2000:49

[CS 2000:38] K-J. Bytner “PPM satsar på egenutvecklat system”, Computer Sweden 2000:38

[DeMarco 1982] T. DeMarco, “Controlling Software Projects. Management Measurements & Estimations” Prentice-Hall, 1982

[Holme 1997] I. M. Holme, B. K. Solvang, “Forskningsmetodik. Om Kvantitativa och kvalitativa metoder”, Studentlitteratur, 1997

[Humphrey 1995] W. S. Humphrey, “A Discipline for Software Engineering”, Addison-Wesley, 1995

[IEEE std 1062-1993] “IEEE Recommended Practice for Software Acquisition”, IEEE, 1993

[Kruchten 1999] P. Kruchten, ”The Rational Unified Process – an introduction”, Addison-Wesley, 1999

[Leveson 1995] N. G. Leveson, “Safeware system safety and computers”, Addison-Wesley, 1995

[Marciniak 1994] J. J. Marciniak, "Software Acquisition", Encyclopedia of Software Engineering, edited by J. J. Marciniak, John Wiley & Sons, 1994, pp. 4-44

[McFeely 1996] B. McFeely, “IDEAL^{SM}: A User’s Guide to Software Process Improvement”, Introduction chapter, pp. 15-24, Technical Report CMU/SEI-96-HB-001, Software Engineering Institute, 1996, http://www.sei.cmu.edu/pub/documents/96.reports/pdf/hb001.96.pdf

[Paulk 1993] M. Paulk, B. Curtis, M. B. Chrissis, C. V. Weber, “Capability Maturity Model for Software, Version 1.1”, Technical Report CMU/SEI-93-TR-24, Software Engineering Institute, 1993.

[Paulk 1997] M. Paulk, B. Curtis, M. B. Chrissis, C. V. Weber, “Capability Maturity Model: Guidelines for Improving the Software Process, Addison Wesley, 1997.

[PPM 99-63] “Tids- och åtgärdsplan för säkerställande av uppbyggnad av kontoadministrativt system”, Premiepensionsmyndigheten, Promemoria 99-63, 1999

[Rational 2001] http://www.rational.com/corpinfo/, Date: May 9, 2001

[SA-CMM 1.01] SEI Carnegie Mellon University, “Software Acquisition Capability Maturity Model, Version 1.01”, 1996

[Sommerville 2001] I. Sommerville, “Software Engineering”, Addison-Wesley, 2001

[SPICE] http://www.esi.es/Projects/SPICE.html, Date: Date: May 17, 2001

[Stapleton 1997] J. Stapleton “DSDM Dynamic Systems Development Method”, Addison-Wesley, 1997

[Stevens 2000] P. Stevens, R. Pooly, “Using ULM: Software Engineering with Objects and Components”, Addison-Wesley, 2000

[Zeithamal 1996] V. A. Zeithamal, M. J. Bitner, “Service Marketing”, McGrawhill, 1996

8.1 Interviews

Date	Name	Company	Project
April 3, 2001	Annika Halldén	Netch Technologies AB	A
April 3, 2001	Per Hökfeldt	Netch Technologies AB	B
April 11, 2001	Lars Rasch	Netch Technologies AB	C
April 17, 2001	Tobias Dahlskog	Netch Technologies AB	D
April 24, 2001	Mats Byback	Sigma Exallon Information AB	E
April 24, 2001	Gregory Urich	Sigma Exallon Information AB	F

A. Appendix A – Cases Of Customer-Developer Interface

This appendix consists of descriptions of six different real world historic customer-developer interfaces in software projects. The structure of each case is as follows:

1. An initial abstract of the project shortly describing:
   - the customer
   - the purpose of the software that was developed during the project
   - some comprehensive characteristics of the customer-developer interface and the development process.
2. A description of the process model of the customer developer-interface
3. A description of the actors and stakeholders and their roles and most frequent interconnections during the project.
4. A description of the most frequent or most important messages identified during the project.
5. An analysis of problems, risks and solutions that can be related to the customer-developer interface during the project.

A.1 Customer-developer interface in project A

A.1.1 Project abstract

The customer of the project was a head-hunting and employment agency. The objective for the project was to develop an Internet-based tool for the labour market with the key features:

- Recruitment advertises and possibility to search for CV’s at the customer’s web site
- Development of customer’s internal organisation and processes
- Possibility to introduce the system as tool for internal recruiting in the organisations of the customer’s customers

A.1.2 The customer developer interface

Different perspectives of the customer-developer interface is described by the following figures and tables:

Figure 46: Project A organisation of increment according to development plan
Figure 47: Actors and roles in project A
Figure 48: Actors in project A and their most frequent interconnections
Figure 49: Identified classes of important or frequent messages between customer and developer in project A
Figure 50: Schematic description of major interface activities during project A

The development project was to be done in four specified increments during a period of two months in late 1999. Each increment was partitioned into four specified and scheduled activities:

- Specification of increment
- Development of increment
- Delivery of increment
- Acceptance test of increment

A single increment of the development process is illustrated in Figure 46 below. The inner organisation of Development - what happens inside the development box - is not considered.

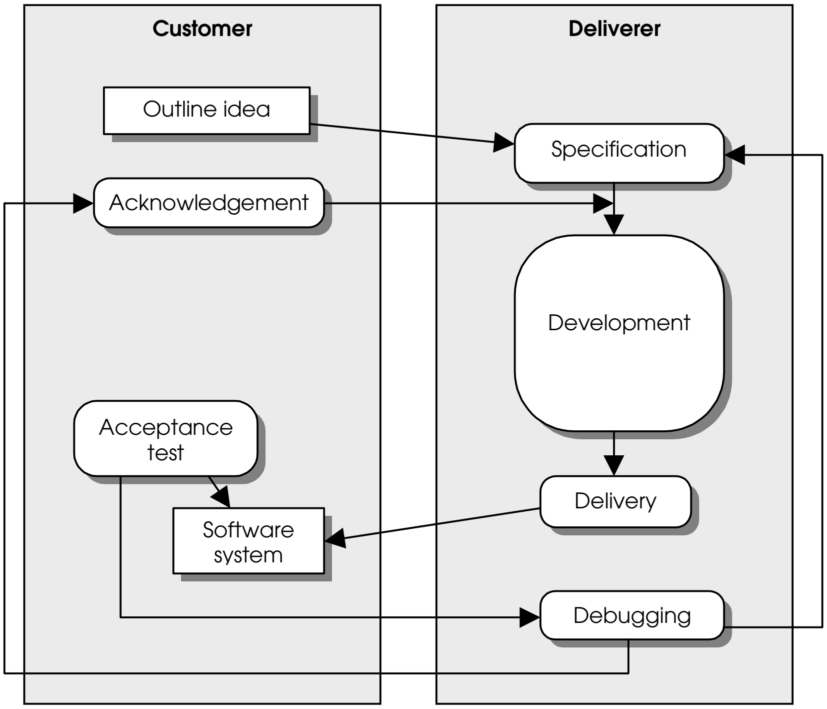


Figure 46: Project A organisation of increment according to development plan

Actor	Role
Customer Project Sponsor	Provide capital and other resources for the customer part of the project on demand from the Head Project Manager Order new features for the system Acknowledge delivery of increment Accept increment on behalf of the customer
Customer Project Manager	Decide on requirements Inform concerned parts at customer side about decisions made with developer Co-ordinate opinions of the Reference Group and the Project Sponsor about specifications and formulate the specification for delivery to Customer Create support for the development process within the Customer Project Group Allocate resources within the customer organisation for the project Communicate customer decisions to the developer
Customer Reference group	Communicate customer needs according work process to Customer Project Manager Schedule work time according to the project schedule made by the Customer Project Manager
Customer Test Group	Make acceptance tests of the increments and report defects and problems to the developer
Layer	Prepare legal actions
Key Account Manager	Preserve a good spirit in the contact to the customer Communicate with the Customer Project Sponsor on major decisions
Project Manager	Be responsible for that the project reaches its objectives Receive orders from the Customer Project Sponsor Communicate to Customer Project Manager Communicate to Development
Development	Implementation System testing
Web Host	Hosting the system after deployment

Figure 47: Actors and roles in project A

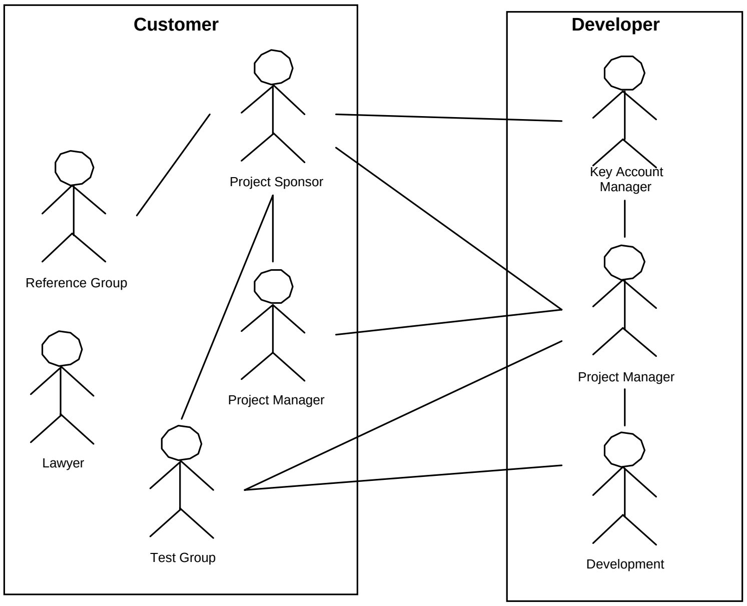


Figure 48: Actors in project A and their most frequent interconnections

Product information:	**Source**	**To**	**When**	**Media**
Increment specification	Project Manager	Customer Project Manager	On start up	Written document
New feature	Customer Project Manager	Project Manager	On demand	By phone, mail or meeting
Increment/feature specification OK	Customer Project Manager	Project Manager	On new feature	Fax
Increment/ feature specification not OK	Customer Project Manager	Project Manager	On new feature	Fax
Request for clarification of requirement	Project Manager	Customer Project Manager	On demand	Phone, mail, fax
Delivery walk through and presentation	Project manager	Customer Project Manager	Before increment delivery	Meeting
Increment delivery	Development	Customer Project Manager		Executable software on the Internet
Increment delivery notification	Project Manager	Customer Project Manager		Phone and mail
Increment delivery acknowledgement	Customer Project Manager	Head Project Manager		Meeting
Problem report on increment	Customer Test Group	Development		Web based form
Debugged increment delivery notification	Project Manager	Customer Project Manager		By mail
Increment accepted	Customer Project Sponsor	Head Project Manager		Fax, mail, phone or meeting
Process information				
Flying (initial Brainstorm)	Customer Project Sponsor	Key Account Manager	Before Project	Conference
Fax delivery notification	Project Manager	Customer Project Manager	When faxing	Phone
Fax delivery acknowledgement	Customer Project Manager	Project Manager	On received fax	Phone
Project Specification	Project manager	Customer Project Manager	At start up	
Project Communication Policy	Project manager	Customer Project Manager	After increment one	
Project meetings	Project manager	Customer Project Manager	Weekly On conflicts	
Project status report	Developer	Customer	14 day intervals	Written document
Event report from	Customer Project Manager	Developer	Weekly irregular on demand	Written document

Figure 49: Identified classes of important or frequent messages between customer and developer in project A

A.1.3 Problem areas related to the customer interface and deviations from plan identified by the developer

Major problems:

- The customer representative in form of the Customer Project Manager did not master the problem domain and did not consult the Customer Reference Group. The consequences of this were that when the acceptance test of the first increment started, the test group found many defects and requested a changed high level design.
- There was no thoroughly gone through, written and signed contract concerning what was to be developed, resulting in conflicts and misunderstandings between the parties through the whole projects. Lawyers where involved twice.
- Customer Project Sponsor ignored, or did not understand the consequences of, signed deals regarding product and process specifications.
- Conflicts inside the customer organisation concerning who was to decide what, when and how led to double messages and inconsistencies.
- The customer developed its business processes in parallel with the software project.

- The only person from the developer that participated in the initial brainstorm (flying) from was the key account manager, who left the project at an early stage. The consequence was that it was only the customer that had a clear picture of the initial vision of the system.
- The development team or the developers project manager was not experienced

Chosen solution:

- After the development of the first increment the parties formulated and signed a communication plan according formal communication and decision rules for the rest of the project. Some, but not al, of the organisational problems at the customer site remained. Figure 50 describes major interface activities during the project.

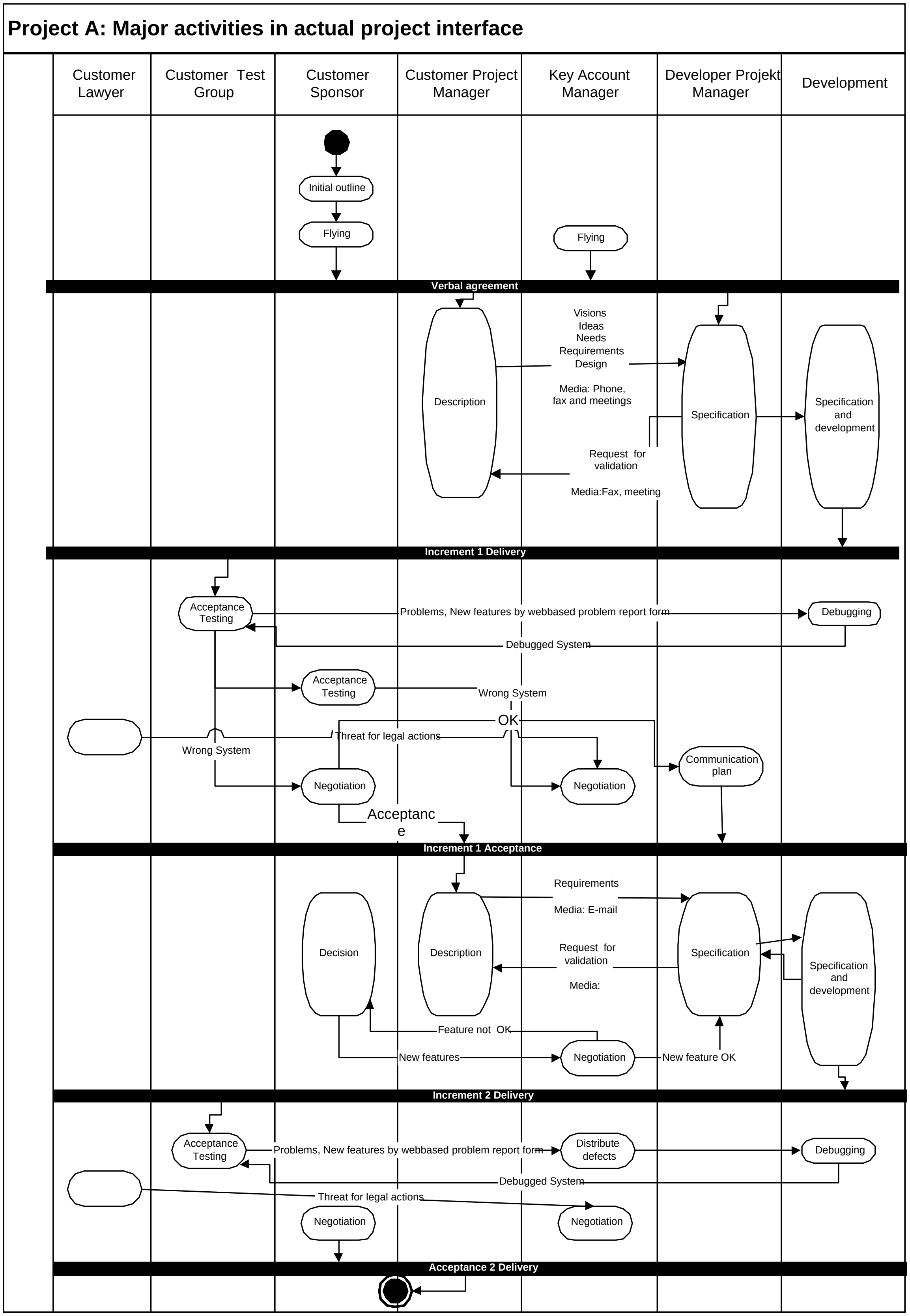


Figure 50: Schematic description of major interface activities during project A

A.2 Customer-developer interface in project B

A.2.1 Project abstract

The customer in the project is a department for internal services at a local site of a global industry company. The purpose of the system is to make company internal services concerning conferences, transports, and locals etc. available for ordering from the other departments (the customers' customer) of the company. The system is divided into seven different head-services. The first phase of the development aims at developing and deploying a common technical and interface platform for the services, together with one of the head services. According to the plan, the subsequent six services will be developed in parallel, in a second phase.

A.2.2 The customer developer interface

The following figures and tables describe different perspectives of the customer-developer interface:

Figure 51: Project B organisation of development of the first service and the technical/graphical platform according to development plan.
Figure 52: Actors and roles in project B
Figure 53: Actors in project B and their most frequent interconnections
Figure 54: Identified classes of important or frequent messages between customer and developer in project B

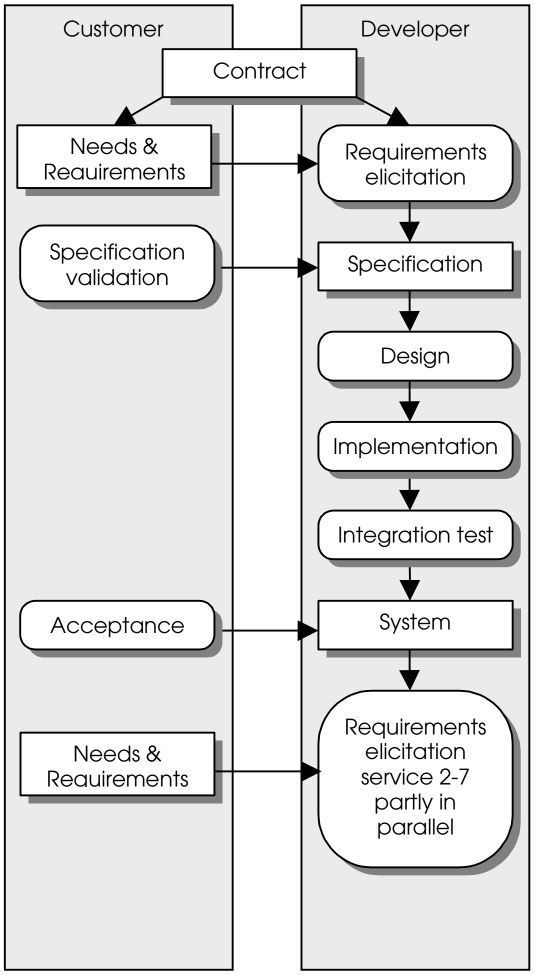


Figure 51: Project B organisation of development of the first service and the technical/graphical platform according to development plan.

Actor	Role
Customer project sponsor	Sponsor of the project Signs contract. Breaks high-level conflicts together with the Key Account Manager.
Customer project Manager	Co-ordinator of sub-projects on customer site. Signs specifications Accepts the system Resolves conflicts within the customer organisation
Service responsible	One sub-project leader for each of the seven head service
Expert user	One responsible for, or expert on, the supply of one of fourteen service component
Customer interface expert	Responsible for graphical and textual quality an constancy
Reference group	The customer's customer, representatives of the end-user
Customer IT responsible	Responsible for the customer's technical system environment
Key Account Manager	Business responsible. Signs Contract. Breaks high-level conflicts together with the Project Sponsor
Developer head project manager	Responsible for day-to-day customer relations, project economy, staffing contract etc
Project Manager	Responsible for technical implementation of the platform and the services. In the second project phase there will be several project managers, each responsible for one service
Development	Specification, Design, Implementation, Integration Test
Head Project manager	External management consultant and advisor for customer and developer in the project paid by both parties Has the casting vote in conflicts between customer and developer
Web designer	External consultant and co-contractor, responsible for user interface

Figure 52: Actors and roles in project B

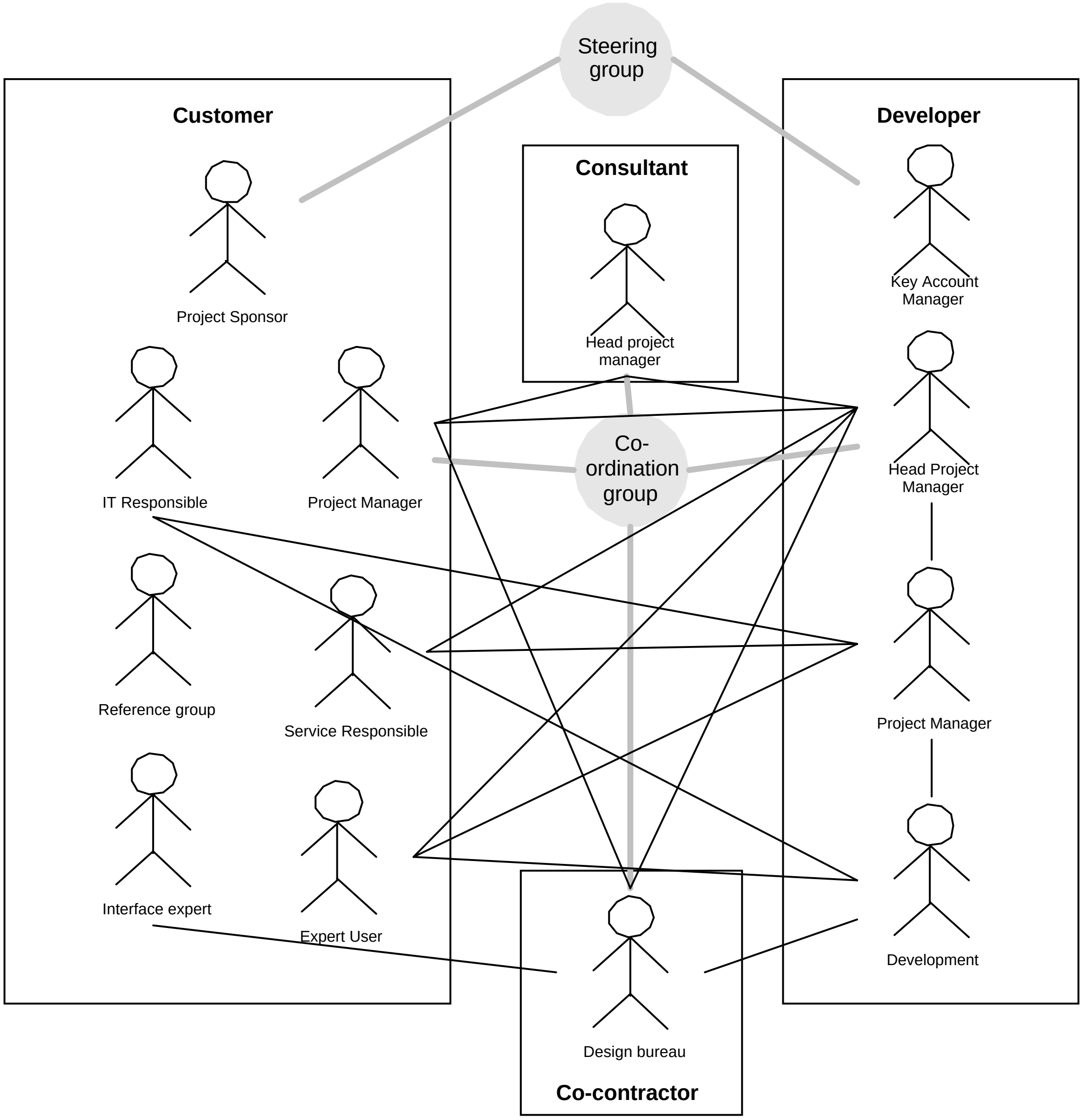


Figure 53: Actors in project B and their most frequent interconnections

Product information	**Source**	**To**	**When**	**Media**
Specification: Use Case	Developer	Customer	On update	Paper media
Specification: User Interface				
Specification: Requirements				
Specification: Dynamic prototype				Live on computer
Request for clarifications of requirement	Developer	Customer		Mail, meeting, phone
Request for validation of requirement	Developer	Customer		
New feature or request for change of feature	Customer	Developer		
The software product	Developer	Customer	Acceptance test	Executable software
Problem report	Customer	Developer	On acceptance	Web form
Release accepted	Customer	Developer		
Process information:				
Project specification			Start up	Paper
Running four week schedule			Weekly	Excel chart
Co-ordination meeting			Friday	
Steering group meeting			Monthly	
Project meeting			On demand	
Release lunch			On release	
Reference group meeting			On demand	
Project Dinners			Quarterly	

Figure 54: Identified classes of important or frequent messages between customer and developer in project B

A.2.3 Problems areas related to the customer interface and deviations from plan identified by the developer

1. **Problem:** The customer did not fully understand the consequences of the agreed specification. When the development team was supposed to implement the system on their own and contacted the customer for smaller adjustments and clarifications, it became clear that the business processes and needs of the customer was not fully understood on neither side.
   Chosen solution: Consequences of this were that the design and implementation phases had to be prolonged and interleaved with continuos and integrated business modelling and requirement elicitation activities. The planned waterfall model converted to prototyping.

2. **Problem**: Specification aspects of written minutes from co-ordination group meetings and e.g. updated use-cases could not be fully understood and validated by the customer. The developer tried to prevent the problem by educating the customer representatives in how to work with use cases. The problem remained.
   Chosen solution: Specifications and prototypes are personally presented to the customer representative.

3. **Problem:** If a customer representative presents an idea, new features or new variants of existing functions to the developer. The developer answers by specifying the idea in form of a prototype, a modified use-case or a textual specification to the customer representative. Frequently the customer representative does not recognise the received specification as a correct interpretation of the initial idea and thus requests a new specification. This scenario may iterate very many times until the developer gets desperate or the customer gets really tired.
   Chosen solution: The developers try to gather all new ideas and suggestions in order to handle them together in a structured way. Planned releases are introduced. Both the customer ideas and the developer's interpretation of the ideas mature over time. Anyhow, the implementation activities will be less disturbed.

4. **Problem:** When a developer needs a simple elucidation of a requirement the formal way of handling this is, in worst case, to go the whole chain: Developer – Project Manager – Head Project Manager – Customer Project Manager – Service Responsible – Expert User. This is costly in terms of time, money and project energy.
   Chosen solution: Developer Head Project Manager gives the Developer and Expert User commissions to resolve the issue on their own (with the risk of consistency problems on a higher system level).

5. **Problem:** Double messages/requirements according the system interface because of limited contact between developer and interface consultant.
   Chosen solution: The interface consultant attends the Friday meetings.

A.3 Customer-developer interface in project C

A.3.1 Project abstract

The customer of the project is a logistic function in a large high technological system development industry. The objective of the system is to let the customer's customer – communication service providers – compose tailored communication systems by selecting, putting together and finally order components from a number of company internal production units within the customer. The production units then dispatch the orders. In short – the system is a business to business web-shop for communication system components.

Initially the customer had planned for only one version of the system. Since then 10 planed releases have been made. The following description and analysis of the customer-developer interface is made a year after the first public system release. In the beginning of the project there were some uncertainties concerning roles and responsibilities of the parties. The relationship between the customer and the developer can now be considered as stable and mature.

A.3.2 The customer developer interface

Different perspectives of the customer-developer interface is described by the following figures and tables:

Figure 55: Project C development process in a standard increment
Figure 56: Actors and roles in project C
Figure 57: Actors in project C and their most frequent interconnections
Figure 58: Identified classes of important or frequent messages between customer and developer in Project C

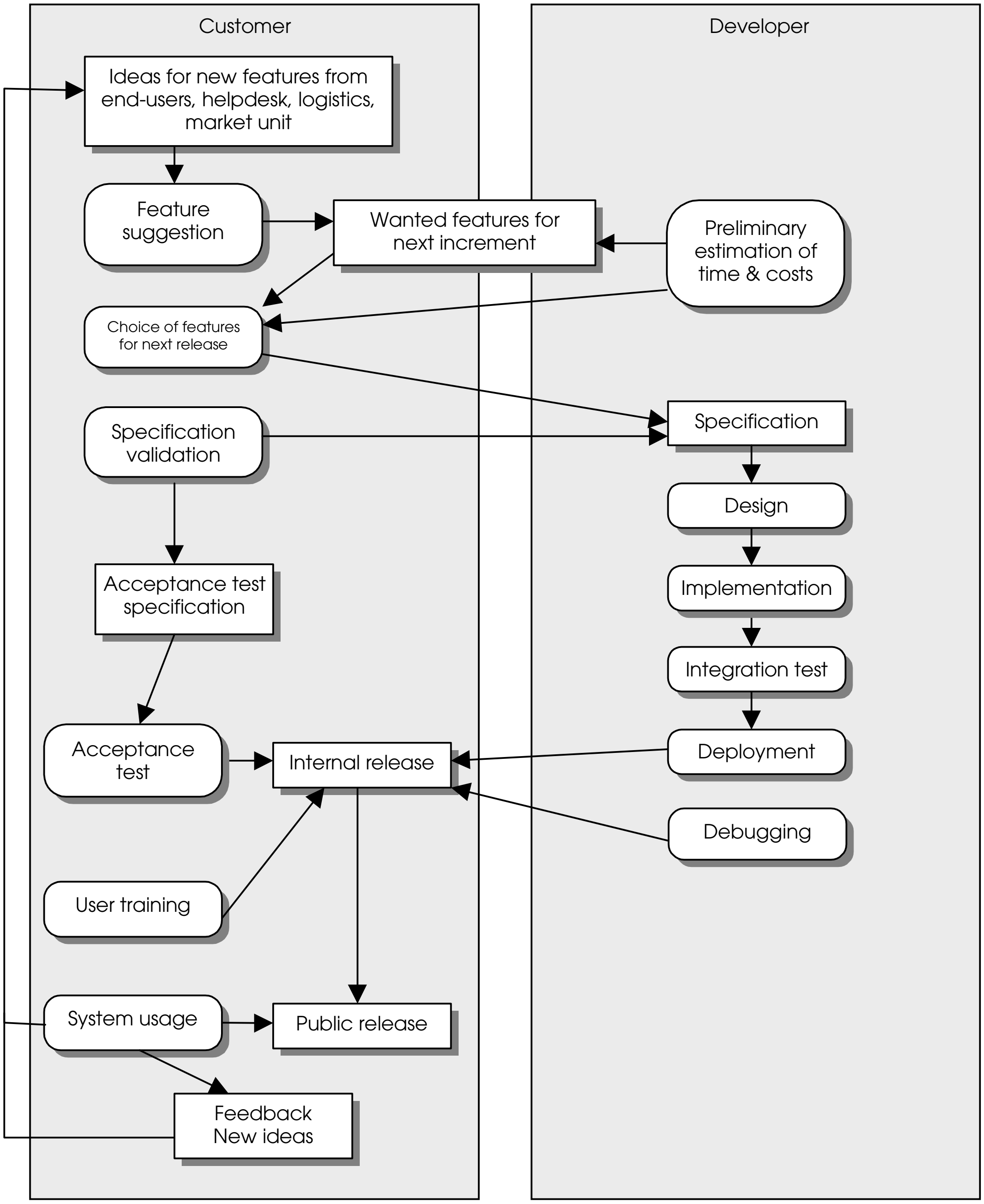


Figure 55: Project C development process in a standard increment

Actor	Role
Customer Project Sponsor	Project sponsor. Signs general agreement. Break high-level conflicts together with the Key Account Manager.
Customer Market Unit	Responsible for contract with customer's customer Accepts system.
Component deliverer	Produces and delivers components and systems to the end-user. Responsible for the internal technical interface of the system.
Customer Project Manager	Co-ordinates customer activities. Breaks conflicts within the customer organisation Negotiates and decides on features and specifications.
User training and manuals	End user training Writes user manuals and guidelines.
End-user	Customer's customer. Web shoppers.
Helpdesk	Assists end-users in how to use the system. Provides ideas for future releases.
Key Account Manager	Business responsible. Signs Contract. Break high-level conflicts together with the Customer Project Sponsor.
Project Manager	Project planning. Time and cost estimations based on feature suggestions. Responsible for technical implementation. Project staffing .
Development	Specification. Design. Implementation. Integration Test

Figure 56: Actors and roles in project C

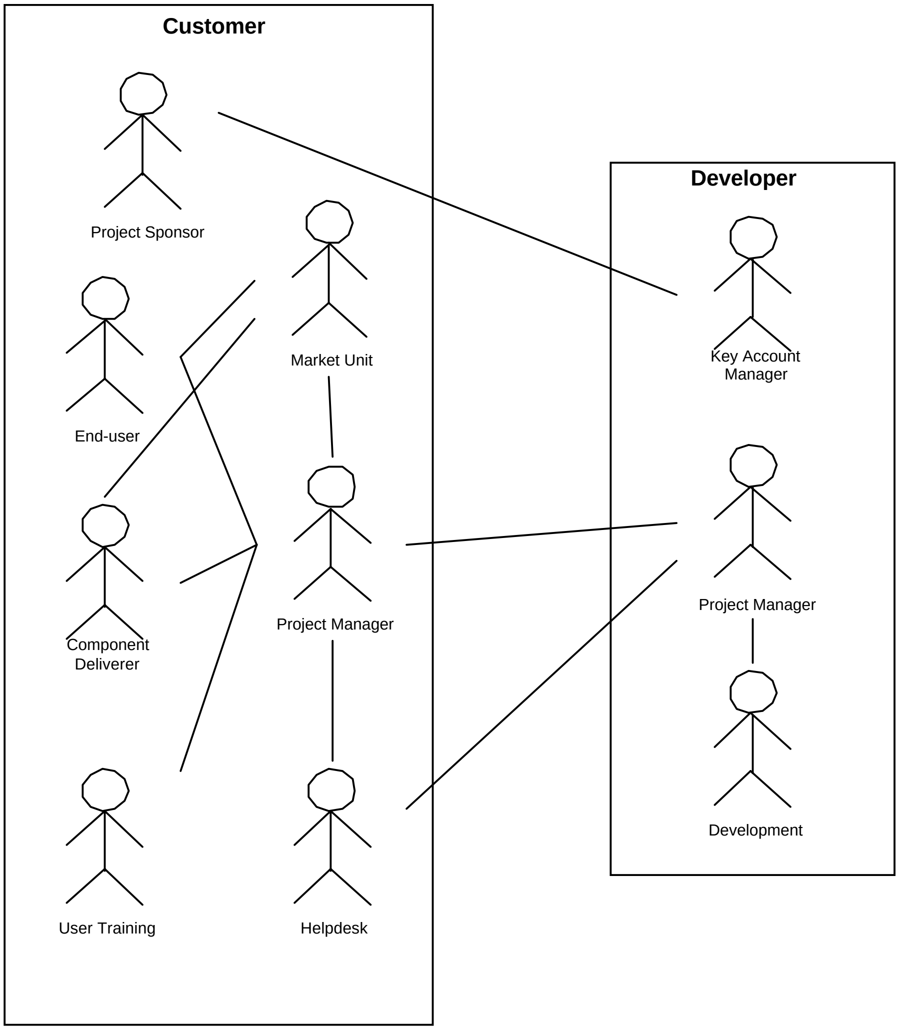


Figure 57: Actors in project C and their most frequent interconnections

Product information:	**Source**	**To**	**When**	**Media**
Feature outline	Customer	Developer	Increment start up	Paper document
Feature specification	Developer	Customer		Paper document
Choice of features	Customer	Developer		Phone, mail, meeting
Request for clarification of feature	Developer	Customer		Phone, mail, meeting
Request for feature validation	Developer	Customer		Phone, mail, meeting
Specification OK/Not OK	Customer	Developer		Phone, mail, meeting
Software product	Developer	Customer		Executable software
Problem report	Customer	Developer		Structured mail
Increment acceptance	Project Sponsor	Developer		Phone, mail, meeting
Process information:				
Feature implementation time & cost estimate	Developer	Customer		Paper document
Increment release date	Customer	Developer		
Project meetings			On demand	

Figure 58: Identified classes of important or frequent messages between customer and developer in Project C

A.3.3 Problem areas related to the customer interface and deviations from plan identified by the developer

Problem: In the beginning of the project, the customer's description of the features described a lot of *ideas for solutions* in favour of *functional requirements*. The customer considered the requirement ready for implementation. This led to a lot of discussions about the form of the customer requirements.

Chosen solution: Successively the customer learned to write more functionality-oriented requirements (What the system was supposed to accomplish) in favour of solution-oriented requirements (How the system should be built). Based on these requirements the developer made implementable specifications of the features. These specifications were then sent to the customer, who after discussion and modifications accepted the specification.

A.4 Customer-developer interface in project D

A.4.1 Project abstract

The customer of the project is a web-shop for consumer products. The project has been running for several years, with e.g. updates of graphical interface, re-factoring and introduction of new features or new categories of articles. In the project there are two co-contractors. One co-contractor is responsible for the hardware platform and to host the system. The other co-contractor is responsible for the database solutions. The database responsible and the developer discuss common technical interfaces and split the development workload between themselves without involving the customer.

Almost all the involved persons at the four parties have worked in the project for several years, knowing each other and the product very well. Some people have changed employer while still working in the project. The developer has exchanged project manager several times. The current project manager has worked with the project for 6 months. During a typical year the system undergoes some minor changes, for instance according new categories of products or graphical re-designs. One or two major releases are also made. For those major releases it is, for marketing reasons, important that they are released at specific dates.

The project can be described as innovative and successful but not structured.

A.4.2 The customer developer interface

Different perspectives of the customer-developer interface is described by the following figures and tables:
Figure 59: Project D development process for a feature or major release
Figure 60: Actors and roles in project D
Figure 61: Actors in project D and their most frequent interconnections
Figure 62: Identified classes of important or frequent messages between customer and developer in project D

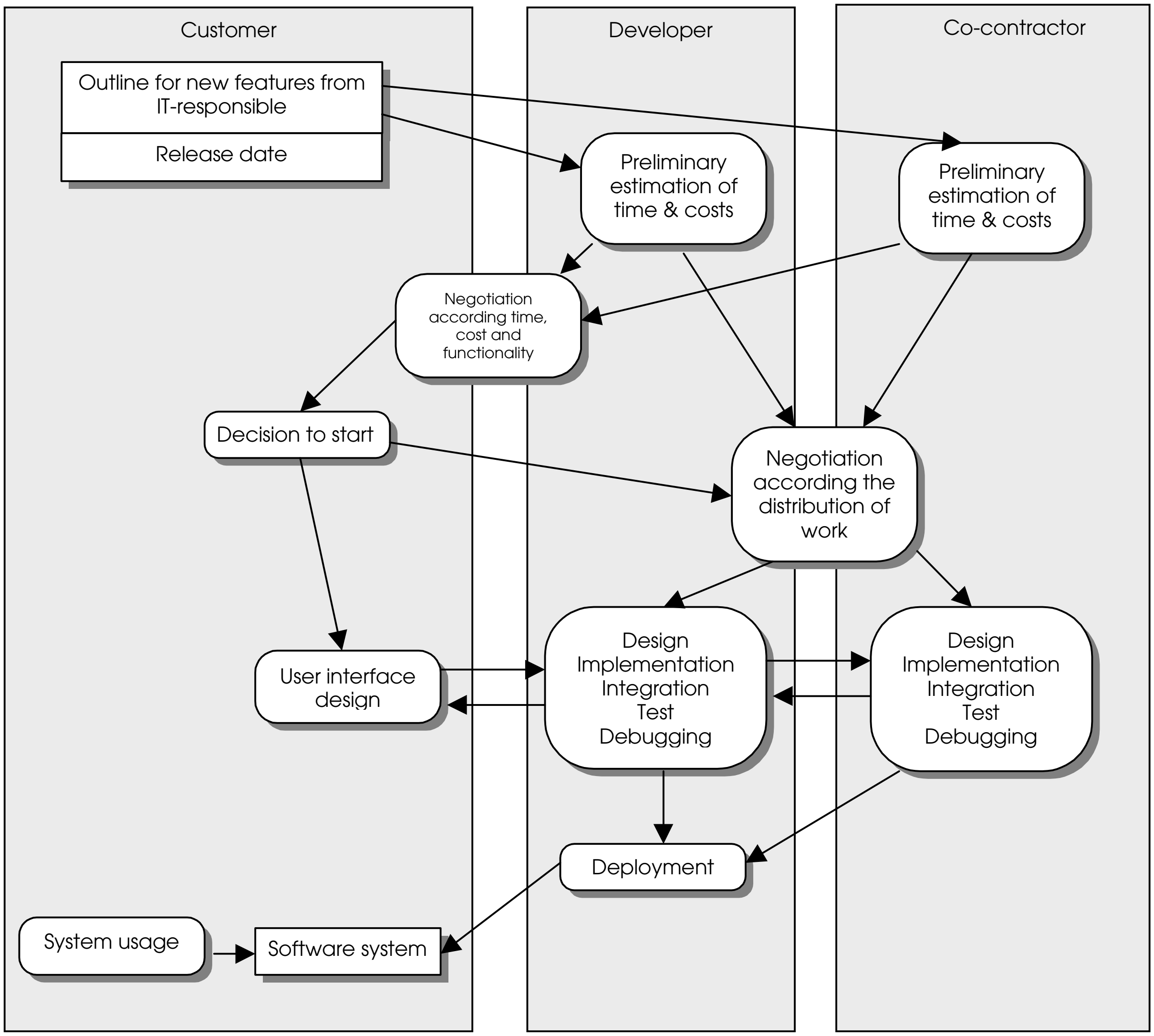

Customer
Developer
Co-contractor
Outline for new features from IT-responsible
Release date
Preliminary estimation of time & costs
Preliminary estimation of time & costs
Negotiation according time, cost and functionality
Decision to start
Negotiation according the distribution of work
User interface design
Design Implementation Integration Test Debugging
Design Implementation Integration Test Debugging
Deployment
System usage
Software system

Figure 59: Project D development process for a feature or major release

Actor	Role
Customer Market Unit	Communicates problems and marketing concept ideas to IT Responsible.
Customer IT Responsible	Project manager at the customer side. Co-ordinates customer activities according the project. Communicates visions and feature outlines to the developer. Negotiates and decides on features and specifications. Accepts a feature or increment.
Customer Web Design	Designs and develops passive parts of the end-user interface.
Customer Helpdesk	Receives feedback from the End-user: customer
End-User: Wholesaler	Delivers consumer products to the customer. Defines a technical interface for product catalogues and product orders towards the system.
End-user: Customer	Customer's customer. Buys consumer products through the system.
Project Manager	Project planning. Time and cost estimations based on feature suggestions. Responsible for technical implementation, project staffing and customer relations.
Development	Specification. Design. Implementation. Integration. Test.
Co-contractor: Data-base	Responsible for design of the customer and product databases.
Co-contractor: Hardware & Host	Responsible for the hardware platform and the hosting of the system.

Figure 60: Actors and roles in project D

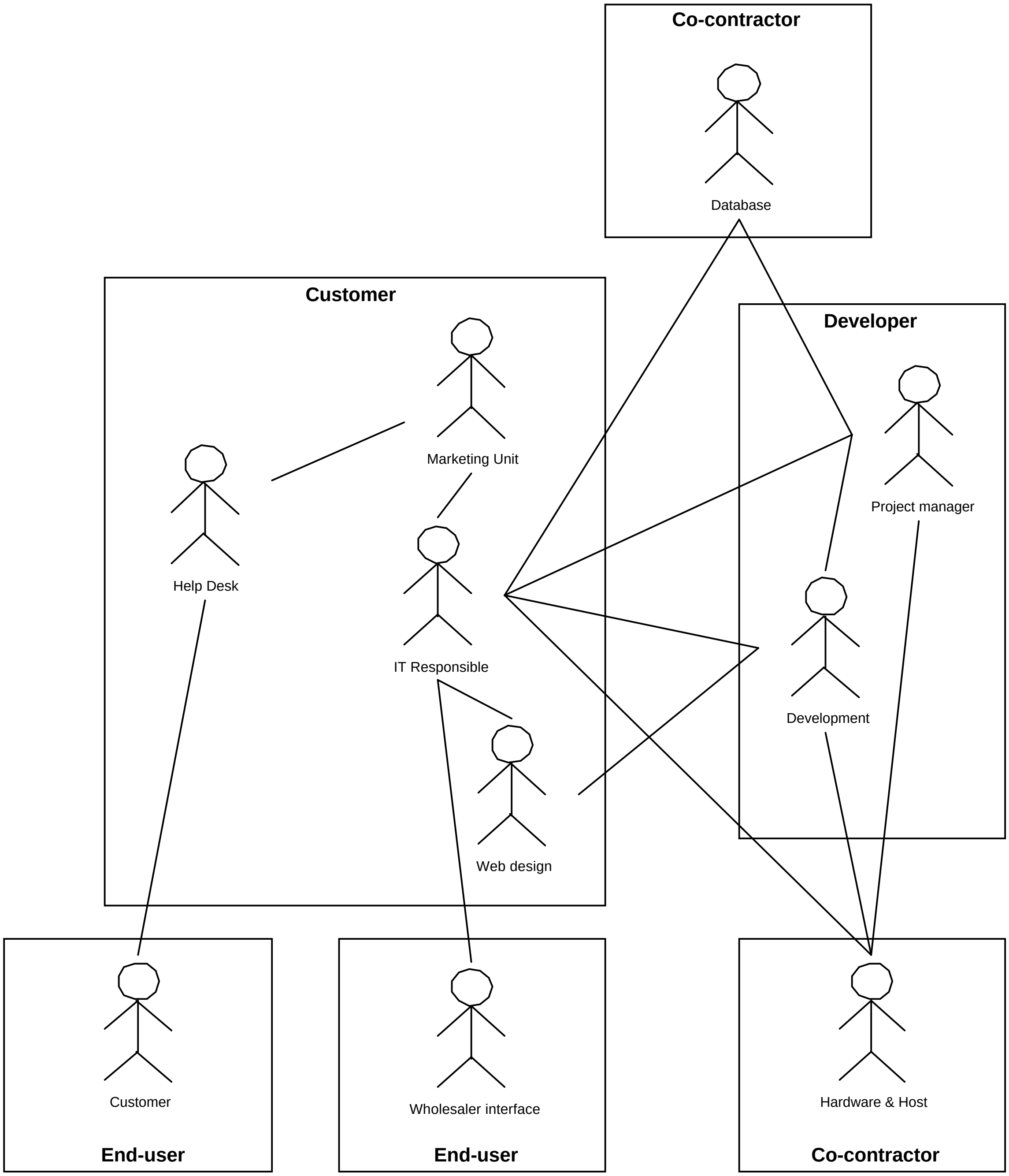


Figure 61: Actors in project D and their most frequent interconnections

Product information:	**Source**	**To**	**When**	**Media**
Feature outline	IT Responsible	Project Manager Development team		Mail, meeting
Problem report	IT Responsible Web design Co-contractor	Project Manager Development team	During development	Phone, mail, meeting
Process information:				
Meeting according project over al status	IT responsible, Project Manager		Monthly	
Meeting according features	IT responsible, Project Manager		Weekly during major releases	
Time report	Project Manager	IT-responsible	Weekly	Mail
Release date	Customer	Developer		Mail
Decision to start development	IT Responsible	Project Manager		Mail
Party	Customer, developer, co-contractor		Now & then	

Figure 62: Identified classes of important or frequent messages between customer and developer in project D

A.4.3 Risk and problem areas related to the customer interface and deviations from plan identified by the developer

Risks: There are several different *risks* of the customer-developer interface of the project that *could have evolved to problems*. Some of the most obvious are listed below:

- In the project there is very little documentation concerning agreements, the project processes or the software system. Most of the documentation that does exist is in the form of sporadic mail messages. This means that the whole system is very dependent of the people that works or has worked with the development.

- During the development of minor features or major releases there has been a lot of communication regarding modifications, clarifications or new features directly between customer representatives and individual members of the development team. If the involved developers were less familiar to the system, this could have caused problems according the consistency of the system.

- In practice, the development team decides in what way an outlined feature is to be interpreted. The influence on both costs and functionality from the customer organisation is thereby reduced.

- Since all three co-contractors work on current account there is no short time incitement to reduce the workload or to plan the development in a detailed way. In the nearness of the technical interface between the co-contractor's applications, double work sometimes is done.

Despite these problematic aspects of the interface, the system works and the customer of the project is successful on its business domain. The main explanation to why the project works is probably that:

1. many of the persons involved in the project have been there since the beginning. They know each other, the system and the business domain of the system very well. The

need for communication regarding what to do and when it should be done is thereby limited.

2. The financial strength of the customer has so far been good. Thereby there have not been any economic incitements for more structured development and communication processes.

Problem: Recently there has been a shift in ownership of the customer, who from now on is a regional part of a global actor. The new owner possesses several different and competing technical platforms with the same main features as the one developed by project D. The new owner is likely to look for possible rationalisations by reducing the number of different platforms. Since the customer still is doing economically well, there is a possible chance for the developer, with or without the co-contractors, to widen the market for the system. This chance would probably be strengthened if the product, the development process and the organisational interface towards customer where better structured.

Chosen solution: The current project manager has introduced a more structured customer interface, contending the following activities and rules:

- All communication with the customer has to pass the project manager.
- The developer tries to transform the co-contractors into sub-contractors in order to reduce the number of communication lines in the project.
- All decisions about economy, features, acceptance and release dates are to be communicated from the IT responsible to the project manager *by mail*.
- On a weekly basis the project manager sends a time report to the customer
- Once a month the project leader and the IT-responsible hold a meeting.

A.5 Customer-developer interface in project E

A.5.1 Project abstract

The customer of the project is a publishing house that saw a market in distributing articles from an encyclopaedia by an Internet subscription service. The starting point was a database of the encyclopaedia articles and the experience of a multimedia version of the encyclopaedia on CD-ROM. The software has several technical interfaces and the development included co-operation with several co-contractors. Some of the technical interfaces were developed in parallel, which made the design and the implementation more complex. Internal users of the system are the article editors and the marketing unit.

The development process followed an instance of Rational Unified Process (RUP). The commitment part of the contract and the functional requirements were based on use-cases. After an initial inception and elaboration phase, the project went through three major increments. Each increment consisted of the implementation of specific use-cases and ended with an installation that the customer could start to use. The last increment ended with an opening of the service.

The project was completed within budget and on scheduled time. After the service was opened for the subscribers, the developer has made three minor revisions of the system.

A.5.2 The customer developer interface

The following figures and tables describe different perspectives of the customer-developer interface:
Figure 63: Development process of project E
Figure 64: Actors and roles in project E
Figure 65: Actors in project E and their most frequent interconnections
Figure 66: Identified classes of important or frequent messages between the customer and the developer in project E

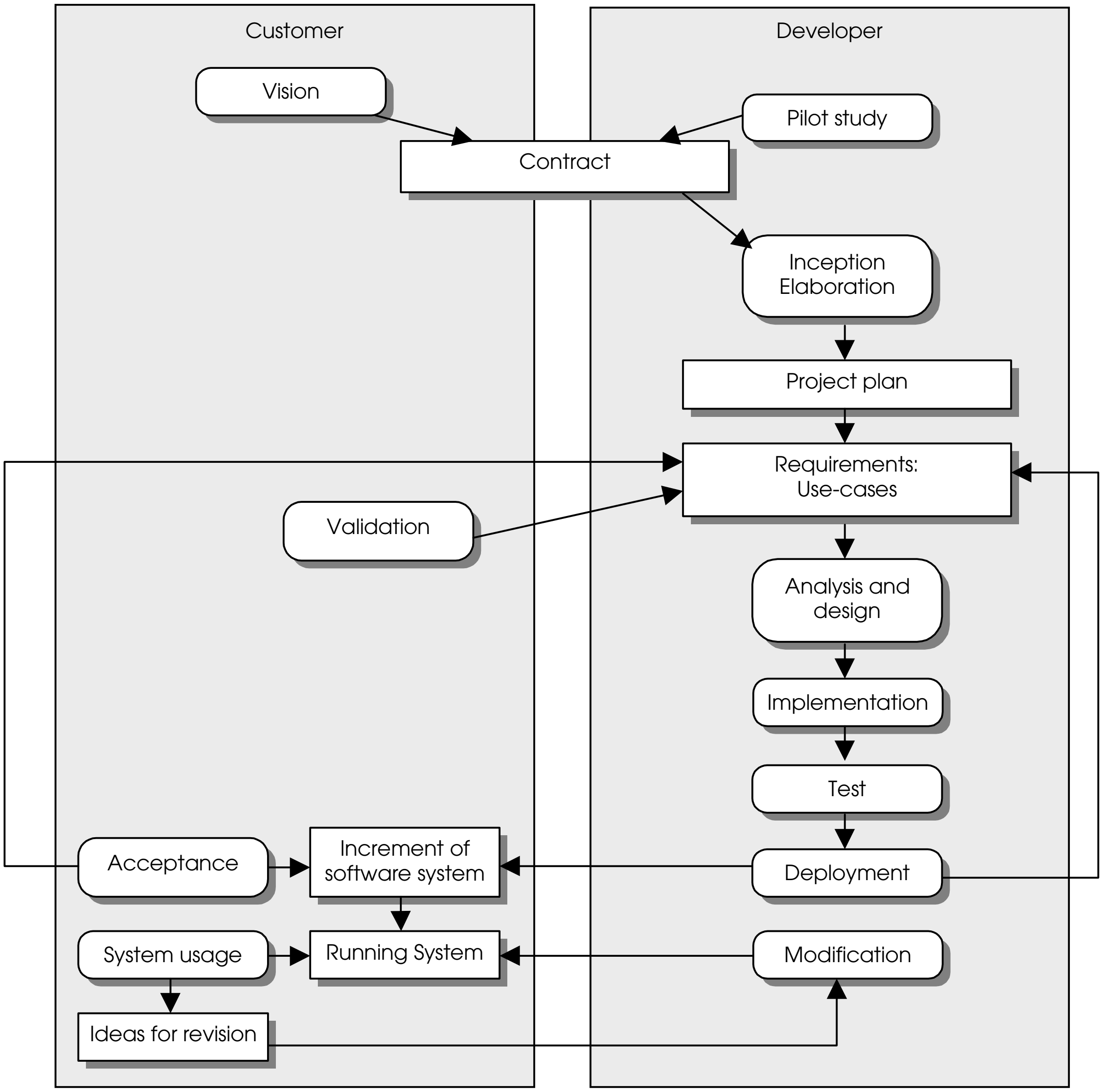


Figure 63: Development process of project E

Actor	Role
Customer Project Sponsor	President of the customer company. Member of project steering group. Project sponsor.
Customer Project Manager	Co-ordinates activities of the customer organisation. Breaks conflicts with CRM System Project Manager
Customer IT Manager	Responsible for customer internal technical infrastructure. Responsible for the acquisition of hosting and thereby the contacts with Co-contractor: Hardware & Host
Customer Market Unit	Responsible for the business goals and the contacts with internal user: Article editors
Customer : Article editors	Internal user, edits articles for the encyclopaedia
CRM System Project Manager	Customer project manager for the parallel development of a CRM-system.
Developer President	President of the developer company. Member of the project steering group.
Project Manager	Project planning. Time and cost estimations based on feature suggestions. Responsible for technical implementation, project staffing and customer relations.
System Architect	Chief system designer. Defines and negotiates technical interfaces to CRM-system, Data Transformation, Multimedia Interface and Web interface
Development	Implementation. Integration. Test.
Graphic design	Graphic design
Usability design	Usability design
Parallel Customer project: CRM-system	
Co-contractor: Hardware & Host	Responsible for the hardware platform and the hosting of the system.
Co-contractor: Data Transformation	
Co-contractor: Multimedia Interface	
Co-contractor: Usability Consultant	Validation of usability.
Co-contractor: Web Interface	Initial graphic design

Figure 64: Actors and roles in project E

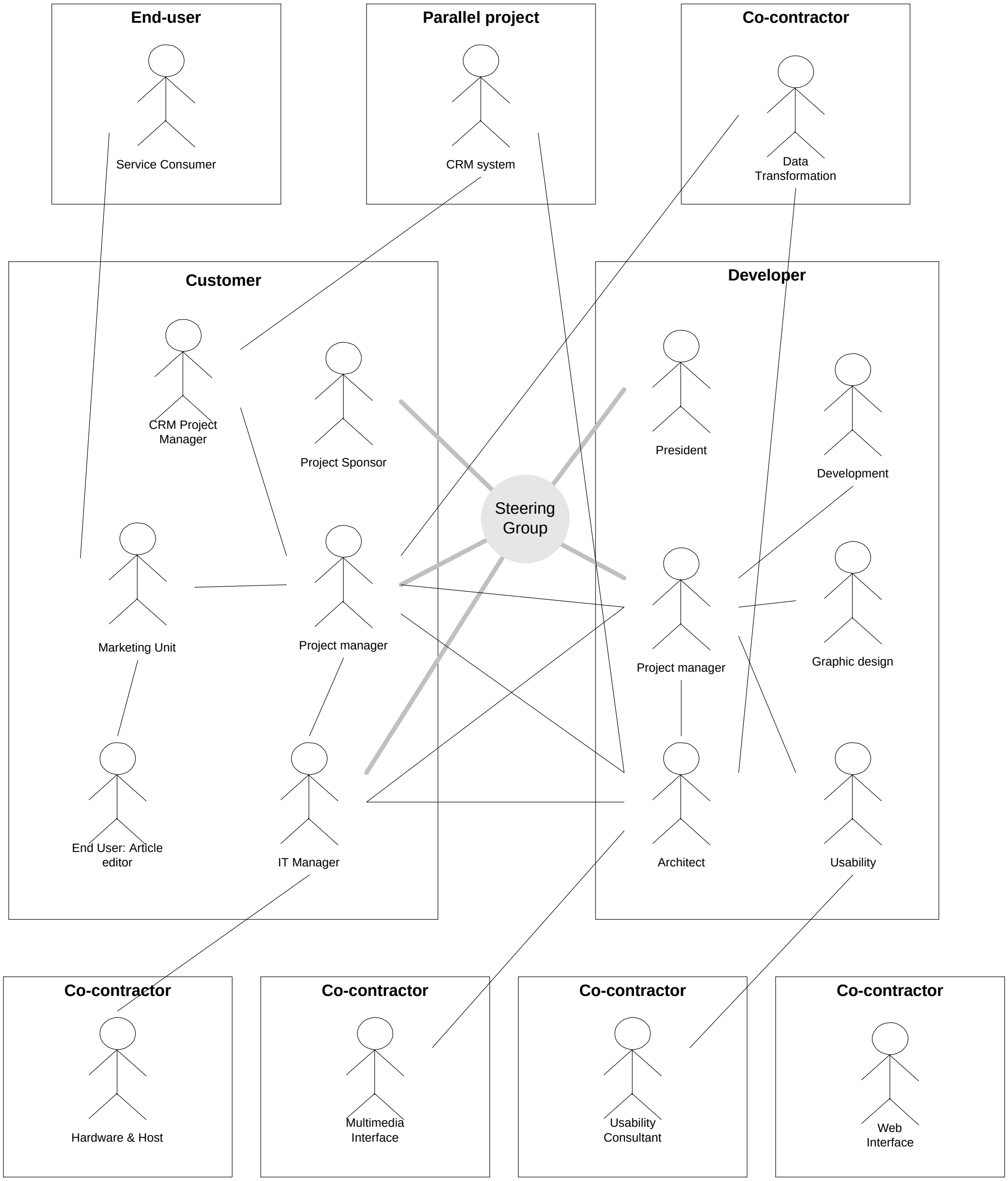


Figure 65: Actors in project E and their most frequent interconnections

Product information	**Source**	**To**	**When**	**Media**
Vision document	Customer	Developer	Initially	
Requirements: Use Cases	Developer	Customer		
Validation	Customer	Developer		
New Features	Customer	Developer		
Change Request	Customer	Developer		Phone Mail Web form
Request for Clarification	Developer	Customer		
Interface Specification	Co-contractor	Customer		
Problem Report	Customer	Developer		Phone Mail Web form
Prototypes				
Increment/feature accepted	Customer	Developer		
Process information:				
Contract	Customer developer			
Steering group meetings				
Reference group meetings	Customer Developer Co-contractors			
Project Meetings			On demand	
Project plan	Developer	Customer		

Figure 66: Identified classes of important or frequent messages between the customer and the developer in project E

A.5.3 Problem areas related to the customer interface or deviations from plan identified by the developer

Problem: The parallel development of the CRM-system and the complications in the technical interface towards the Data Transformation System, led to complex negotiations according functionality and solutions. The communication with the co-contractor responsible for the Data Transformation System could be managed directly between the co-contractors. The communication regarding the CRM-system sometimes had to pass through the project managers for the respective system at the customer side. A practical consequence of these problems was that the implementation of some of the use-cases had to be postponed to later increments.

Problem: In the early party of the project the developer tried to make the customer communicate problems, change requests and new ideas through a structured word-form. The messages that arrived in those forms was not very structured and had to be complemented by phone or mail anyway.

Chosen solution: In the later part of the project the structured word-form was exchanged to more informal excel-chart. The developer took a greater responsibility for structuring the contents of the messages.

A.6 Customer-developer interface in project F

A.6.1 Project abstract

The customer of the project is an Internet site for a large and well-defined target audience. The activities were to be financed by commercial advertises. The product to be developed was a web-publishing tool with features for treating memberships, advertisements, chat forum, a flexible graphical design etc.

After a pilot study the developer found that the desired system would cost more than the customer could afford. Since the technology to be used was partly new to developer and the developer saw possibilities for re-use of vital parts, the developer agreed to run the project at an economic loss. This loss became heavier than planned.

Initially the development was planned to be done as a pilot study followed by three increments. Delays in the project reduced the number of increments to two.

A.6.2 The customer developer interface

Different perspectives of the customer-developer interface is described by the following figures and tables:

Figure 67: Project F development process
Figure 68: Actors and roles in project F
Figure 69: Actors in project F and their most frequent interconnections
Figure 70: Identified classes of important or frequent messages between customer and developer in project F

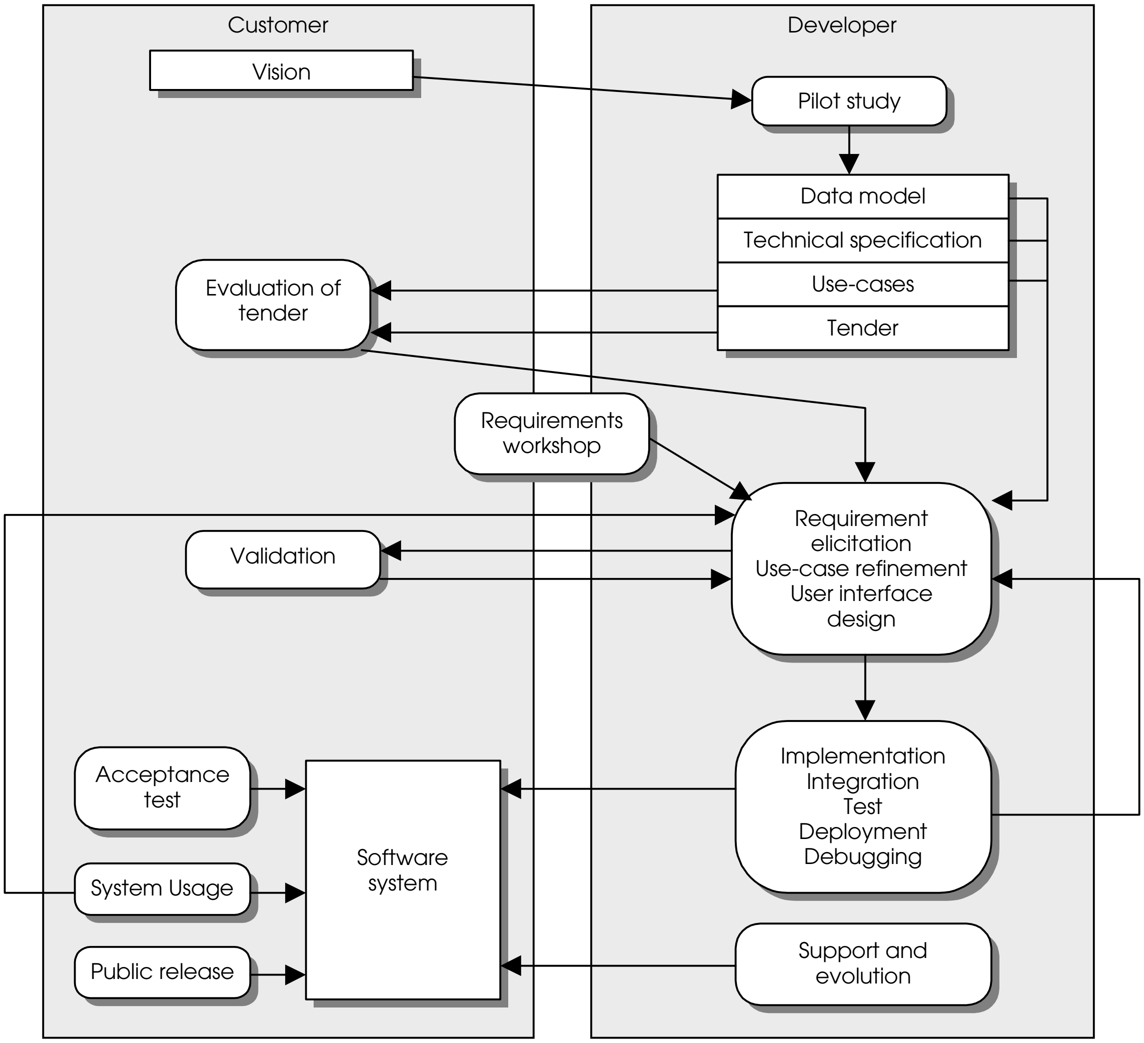


Figure 67: Project F development process

Actor	Role
Project Sponsors	Finances the project by contributing venture capital.
President of the customer company	Project Manager at the customer side. Chief editor of the web site. Only customer actor in the first increment. Employer of the internal users.
Customer Internal users	Creates and publishes material for the web site. Administrates the web site. Runs acceptance test during the learning of the system after the deployment of the first increment. Report problems.
Customer Responsible	Creator of the tender.
Project Manager	Project planning. Responsible for the user-interface. Responsible for technical implementation, project staffing and the relations to the customer president.
Development	Specification. Design. Implementation. Integration. Test.
Advertising Agency	Initiates the initial contact to the customer. Outlines the graphical design.

Figure 68: Actors and roles in project F

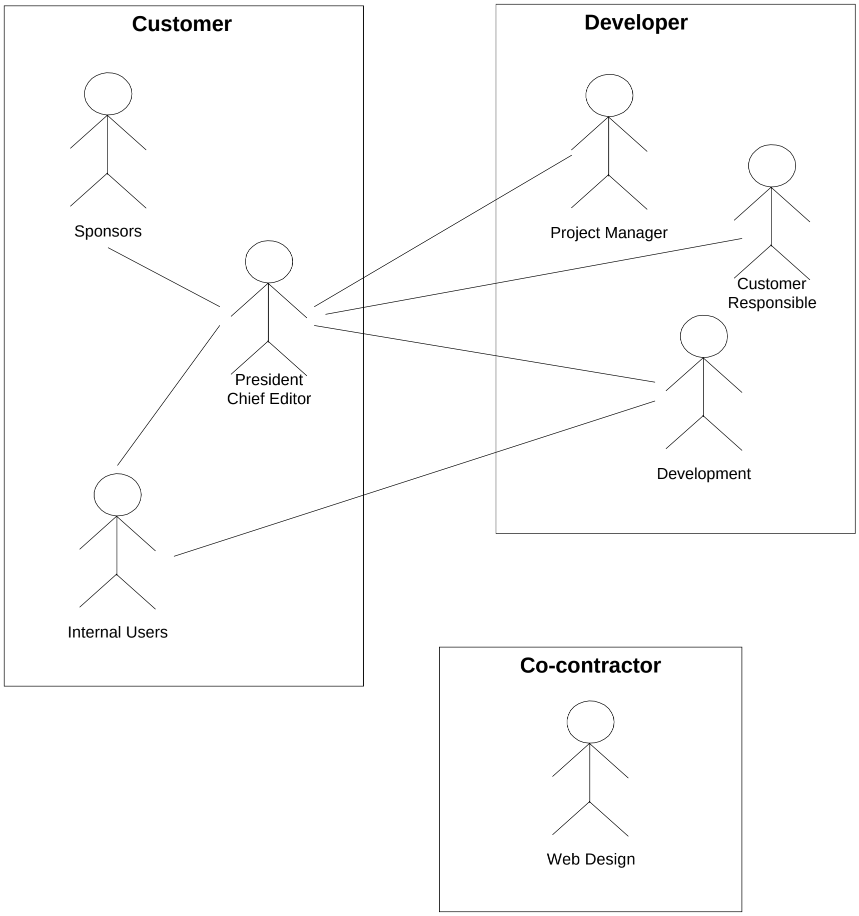


Figure 69: Actors in project F and their most frequent interconnections

Product information:	**Source**	**To**	**When**	**Media**
Vision	Customer	Developer	Initial	Textual and oral description
Requirements workshop	Customer Developer		Requirements elicitation	Meeting
Use-case	Developer	Customer		Textual description
Interface prototype	Developer	Customer		Paper illustration
Prototype walk-through	Developer	Customer	Use-case validation	Paper illustration and meeting
Use-case OK	Customer	Developer		
Problem report/change request	Customer	Developer	Acceptance test System usage	Excel-chart, mail, phone
Increment OK	Customer	Developer		
Process information:				
Tender	Developer	Customer		
Project Specification/Plan	Developer	Customer		
Project meetings			On demand	

Figure 70: Identified classes of important or frequent messages between customer and developer in project F

A.6.3 Problem areas related to the customer interface and deviations from plan identified by the developer

1. **Problems:** The customer changed and evolved the requirements during the development of the first increment, at the same time denying doing so. The project manager got partly stuck between the joint-risk-characteristic of the project and the need to charge extra for new features.

   Chosen solution: The cost of implementing new features was split between the parties.

   Suggested solution: Train the customer representative in the consequences of the decided development process as an attempt to give a picture of which problems that probably will arise at different phases of the project and how those problems best can be handled.

2. **Problem:** The developer tried to make the customer communicate new features, change requests or problems via a structured excel-chart. This formal approach did not work since the customer often bypassed the excel-chart by mail or phone. Alternatively the customer was not capable of describing the problem/change in a form that was useful for the developer.

   Solution: Frequent discussions regarding features, use-cases and prototypes.

3. **Problems:** Lack of developers in combination with evolving requirements resulted in a two weeks delay of the first increment

   Chosen solution: Argue that the delay was caused partly by customer-evolution of the requirements and therefore to some extent compensate the delay with extended functionality.

4. **Problems:** The customer did not fulfil the payment schedule. In fact the customer did not pay at all. This economical aspect of the project was not discovered until the project was in the middle of the second increment. The basic reasons for the delays were probably to find in conflicts between the external project sponsors and the customer.

   Chosen solution: Since the developer had a self-interest in the project and since one of the external sponsors also had a direct owning interest in the development company the project was finished and the system successfully released to the Internet. Until payment is settled, the developer will not give any support to the customer nor participate in any system evolution.