Chapter 1
What We Say We Want and What We Really Need: Experiences on the Barriers to Communicate Information System Needs

Aapo Koski and Tommi Mikkonen

Abstract Information system requirements are meant to communicate the relevant needs and intention to a wide range of stakeholders. The requirements form the basis on which the tenders are issued, projects are agreed upon, and service-level agreements are made. However, as the requirements state what the system owners—or the ones who are willing to pay for the system—want the system to achieve, they reflect the owners' views and understanding. This setup is plagued by many weaknesses. First, the system owners are seldom experts in the information system design and therefore they may be unable to state all the relevant requirements comprehensively. Second, no matter how much energy and time is invested in the requirement definition and elicitation, many aspects of the requirements are only revealed during the development and deployment, and remain unforeseen until later on, when the development is well under way. Finally, the required system architecture cannot be appropriately designed, if we do not know the requirements at a sufficient level. In this chapter we reflect our experiences from participating in a number of large, commercial information system development projects in both public and private sectors in which the traditional way of handling the requirements has proven to be insufficient. With the software as a service (SaaS) business model, where the goal is frequent releases and continuous delivery of ever-improved services, the associated weaknesses become even more prominent. We propose better practices for specifying systems and suggest concentrating a lot more on the true communication and discussion, focusing always on the most important issues and most important stakeholders only and keeping the vision updated and clear for the whole duration of a system development project and also the system maintenance period.

A. Koski (✉)
Basware Oyj, Tampere, Finland
e-mail: aapo.koski@iki.fi

T. Mikkonen
Tampere University of Technology, Tampere, Finland
e-mail: tommi.mikkonen@tut.fi

M. Ramachandran and Z. Mahmood (eds.), *Requirements Engineering for Service and Cloud Computing*, DOI 10.1007/978-3-319-51310-2_1

Keywords Requirements specification · Requirements management · System architecture · External quality · Internal quality · Process quality · Communication · Software as a service · Continuous delivery

1.1 Introduction

When referring to requirements related to an information system, we usually mean requirements originating from customer-side stakeholders. These requirements represent the views and needs of the people at the business or enterprise operations level, covering end-users, acquirers, customers, and a number of other stakeholders. The requirements are recorded to solve the stakeholders' problems. These problems to be solved vary in their nature, and often the different stakeholders perceive the situation in very different ways. To further complicate the matter, the set of requirements for fulfilling stakeholders' needs typically has a strong connection not just to the domain the stakeholders represent, but also to the history and to the environment in which the system is to be operational.

In addition to stakeholder's needs, we have system requirements, which should unambiguously specify how the system is to be developed as well as define the basis for system architecture design and technology selections. System requirements are naturally closely related and connected to the customer-side requirements, but they ought to be written in a language that is fully and unambiguously understood by system designers, architects, developers, testers, as well as other people that work on the system provider's side. Ideally, system requirements complement the original end-user requirements and provide deeper understanding on what the system needs to be capable of doing. Making stakeholder and system requirements meet—so-called requirements elicitation—appears to be the hardest and the most critical part of the development of any larger information system. Furthermore, the situation gets even more complex as new boundaries emerge over time, for instance when the business model changes. Today, the most dramatic change is when installable systems operated in full in customer premises are being replaced by software that is provided as a service.

This chapter reflects industrial experiences gained from a number of large, commercial, software as a service(SaaS) information system development projects for both public and private sectors during the last decade. As we, the authors of the chapter, represent the system providers, we look at the information system projects from the system provider's point of view, from which it is not always quite clear where the customer-side requirements originate, what kind of a history the requirements have and what kind of real needs are hiding behind the stated requirements. In particular, we address the dependencies and overlaps between requirements, which require extensive effort to understand properly, but if left not fully understood, pose a significant threat to the success of the system development project. While a prominent attribute throughout the information system projects has

been that the bigger the project, the larger the number of requirements is, it is equally clear that the traditional way of handling the requirements has proven inadequate in several respects. Having had the first-hand opportunity to observe this has thought us valuable lessons for later use.

The rest of this chapter is structured as follows. In Sect. 1.2, we discuss the characteristics of the requirements and their diverse role in the information system domain. In Sect. 1.3, we provide an insight to the actual requirements we have encountered in the projects and discuss the possible reasons why the requirements often are like they are: far from optimal. In Sect. 1.4, we present our understanding on the nature of the true needs of the system owners and how the true needs are reflected in the written requirements. In Sect. 1.5, we discuss the barriers to communication which, based on our experiences, are the main reason for low-quality requirements and requirement management. In Sect. 1.6, we discourse the relation between the requirements and the system architecture. Finally, in Sect. 1.7, we draw some final conclusions and present better ways to collaborate and communicate, thus enhancing the quality of the requirements.

1.2 The Challenging Role of the Requirements

With requirements, we aim at specifying what a computer system or service should do [1]. The requirements are important in a number of ways in the development [2], including typically at least the following views:

- They form the basis of the system requirements and various activities related to the system design;
- They form the basis of system validation and stakeholder acceptance;
- They act as a reference for all the integration and verification activities;
- They serve as means of communication between all the stakeholders, like the technical staff, management, and the stakeholder community.

Based on our industry experiences, especially the final view is often overlooked and misinterpreted. Yet the role of the requirements as a means of communication to a vast audience is pivotal in any development effort where contracting plays a role, because the requirements form the basis on which the tenders are issued [3], projects are agreed upon [4], and service-level agreements are made [5]. Altogether, the requirements are fundamental to the economic context in the software development: if the quality of the requirements is not good enough or the requirements are not managed properly, the projects' risk to fail increase significantly, the total cost of ownership of the systems increase, and the overall user satisfaction will not be as good as could have been expected. As reported in, e.g., [6, 7], low-quality requirements are the most common reason for the failure of software projects.

The setup where a predefined, fixed set of requirements to satisfy the user needs exists has many weaknesses [8, 9]. Firstly, the people involved in the requirement

creation, the system owner and other stakeholders, are seldom experts in the information system design and are thus unable to state all the requirements—especially the nonfunctional ones—with enough detail and rigor to be unambiguous and comprehensive. Requirements that remain ambiguous will be misunderstood, their effect will not be reliably estimated, and they cause a lot of waste in the form of lost time and resources as they are re-specified, detailed and unfortunately also expanded many times in course of a project. Secondly, no matter how much energy and time is put into the creation of the requirements, for any larger information system, many of the true requirements are only fully revealed during the development and deployment of the system itself [10]. Finally, the system architecture, which has the important role to describe how the system will be organized and therefore also to define what the system can or cannot do, cannot be appropriately designed if we do not know the requirements at a sufficient level. Especially, if architecture design is based on assumptions made solely by the system provider side designers and architects, it will be highly unlikely that the architecture will behave in a way that fulfills all the true customer-side requirements.

An important factor that should not be forgotten is that requirements are not only limited to the functionality of the system, as often supposed, but include other aspects as well. Different authors have presented various definitions, but in general functional and nonfunctional (or quality) requirements are separated. For the purposes of this chapter we may take into use the classification by Davis [11]:

- *Functional requirements*: These include statements regarding the services which the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- *Nonfunctional requirements*: Constraints related to the services or functions offered by the system, such as timing constraints, security issues, constraints on the development process and so on.
- *Performance and reliability requirements*: Requirements that specify the speed or operational effectiveness of a capability that must be delivered by the system architecture.
- *Interface-related requirements*: Requirements that come from the environment of the system and that reflect characteristics of the integrability of the system.
- *Design constraints*: Requirements that come from the application domain of the system. Typically, such reflect all kinds of constraints that need to be taken into account in that specific domain.

In addition, there often are domain or system specific requirements, for example associated with applicable standards or laws. For instance, developing medical devices or aerospace systems typically have special requirements that are associated with the development of the system rather than the eventual outcome. The problem here is that a service provider cannot always trust that all the relevant domain and system-specific requirements are stated by the system owners as the original requirements that need to be fulfilled and taken into account. This aspect is further emphasized in the SaaS environment, as the end-users who may or may not be

aware of the laws and standards, not necessarily verify that the service provided is standard compliant or legal. The unawareness of the end-users, however, does not naturally acquit the service provider from the responsibilities.

1.3 What We Say We Want and Why

To identify possible reasons why the requirements are as they are, we next look at what the stakeholders' requirements of a large-scale information system specification typically contain and what the typical process resulting in the specific set of requirements is. In the scope of experiences presented here, all the projects have started from the system providers' point of view with a request for proposal (RFP) or request for quotation (RFQ) process [12, 13]. While both mark a request for describing a project and sometimes the difference is only in the eye of the beholder, the principal difference between the RFP and the RFQ processes is that commonly the RFP is used when the stakeholders concentrate on describing the problem they have and want to hear in which ways it should be solved. In contrast, the RFQ is used when the stakeholders know what they want but need information on how potential vendors would meet the requirements and how much it will cost. The goal of both the RFP and RFQ is that the potential suppliers are informed that certain service is needed in a way that enables the suppliers to respond factually to the identified requirements. Often, prior to the issuing of the RFQ or RFP to the potential providers, a round of the standard business process of request for information (RFI) [14] is also executed, where the potential vendors are asked to determine what kind of services are available to meet the described needs.

Both RFP and RFQ processes of information systems, disregarding if their domain is public or private sector and also almost regardless of the target and characteristics of the system itself, have taken the habit of starting with a comprehensive specification phase, possibly including also a round of a RFI process. In this phase, the representatives of the future owner of the system (and often also a hired group of consultants) spend a considerable period of time trying to determine the exact needs of the owner as precisely as possible. Typically, the specification phase results in hundreds or even thousands of requirements, written carefully in, e.g., IEEE 830 requirement format [15]. These requirements are reviewed many times, over and over again, and usually they are finally accepted by all the stakeholders, although usually only after a number of refinements and iterations. However, for the potential system providers the details of these processes remain mostly unknown, due to the common reckoning that by giving any of the potential system provider information on the work-in-progress would introduce unfairness in the tender to build the system based on the RFP or the RFQ.

Nevertheless, after the competition based on the RFP or RFQ has been decided and an agreement on the development of the system is signed with a company, this defined, refined, iterated, and finally accepted set of requirements serves as the most valuable artifact for the following steps in the information system design and

development process. The requirements are treated in various occasions as the final "truth" to which one can safely refer to when discussing system features or characteristics and by which the customer assesses the progress and the quality of the created system. As anyone involved with information system design and development can verify, the fixed set of requirements is already a problem in itself, and, if not adjusted and refined in a continuous manner, leads often to system implementation that does not correspond to the actual needs of the system owner. Moreover, even if the requirements itself were at some point of the system development process complete and able to reflect the true needs of the system owner, the form and the language used in the writing of the requirements leaves typically a lot of freedom to the interpretation of the requirement. This problem is emphasized when the project is large in size, related to multiple organizations and lasting for a long time, because then the same requirements can be interpreted differently in different phases of the project, when considered by different people playing different roles in the development.

The requirement gathering and specification phase commonly overlooks the architecture of the system to be developed. Although some of the requirements can be categorized as architecturally significant [16], the main features and required characteristics of the system architecture largely remain unspecified and vague. Especially at the present era of agile development methods, where room is typically left for later fine-tuning and revisions, requirements that are specified upfront in the project easily lead to a so-called emerging architecture [17]. Emergent architecture in itself sounds acceptable—and indeed is acceptable for various projects—but for long-lasting, dependable systems and as well as for systems provided by SaaS model, such an approach means that the system architecture has to be rethought many times during the project [18] as we learn more of the system we are developing. This in turn costs money, makes the system development slower, limits the options, and causes a lot of confusion as several, incompatible views to the architecture are simultaneously under consideration by various stakeholders.

The requirements, the architecture, and the original needs or vision of the system to be developed and taken into use give us three points of view into the system. These views are not necessarily similar, however. Ideally, these three elements, complemented with the acceptance testing criteria or test cases, should support and complement each other and together give any stakeholder a clear view on what are we developing and how well it satisfies the needs we have.

When specifying the requirement in the fashion described above, we make, partly unconsciously, partly fully consciously, assumptions, some of which are almost too bold, namely:

- The representatives selected or set to specify the system have all the knowledge required to do the specification job.
- The understanding on the needs we have will not change during the specification process.

Table 1.1 Examples of functional and nonfunctional requirements in industrial projects

Requirement	Type	Comment
"The system shall provide the user the time she has been handling her task"	Functional	Requirement does neither specify what is meant by the time nor give any indication why the time is important to the user
"The system shall support the addition of new ways to communicate"	Functional	Requirement does not limit in any way what kind of ways of communication should be supported and what supporting really means
"The system shall support high availability"	Nonfunctional	Requirement is far too generic and does not succeed in communicating any true need to the reader
"The system shall present all the required information stored in the system to the user in real time"	Non-functional	Requirement cannot be truly met in any distributed information system as there always are delays in the transferring, handling and visualizing the information. The requirement does not give any indication in what scale the system needs to be a real-time system

- The representatives are capable of expressing the needs in a written format that cannot be understood in a wrong or incomplete way by other people and the written down needs leave as little space as possible for vagueness.
- The requirements can be written in the same way regardless of whether a requirement specifies a functional feature or a quality issue.
- The requirements are expressed in such detail that the potential vendors can reliably estimate the time and costs required to implement each requirement.
- Since information systems are nowadays created in an agile—although the most appropriate term in connection with tender projects is agilish—way, the requirements need to be written in a way that allows incremental development. Ideally, the requirements should thus be independent of each other, and therefore implementable in any order.
- Related closely to the previous assumption, the requirements can be written in a way that allows iterative development: the implementation of a feature related to a requirement can be tested, verified, and accepted after an iterative development phase (like sprint), although the need that the requirement reflects is not necessarily fully satisfied by the iterative step in question.

To highlight the points given above, let us take a look at some real-life requirements[1] that we have encountered. These requirements demonstrate the typical qualities and unambiguousness of the requirements (Table 1.1). As can be clearly seen in the examples, even if elaborated with the system owner in project

[1]Requirements are taken out of their context to protect the identity of the system in question.

negotiations or other such meetings, the requirements are easily understood differently by different stakeholders, and the estimates on the effort needed to fulfill the requirements are next to impossible to make reliably. The original system owner's intention and the interpretation of the requirement by a developer can easily be totally two different things.

Furthermore, it seems that the bigger the program is and the longer the specification period has been, the greater and bolder the assumptions are, and, consequently, also the greater are the risks involved. This at least partly false feeling of certainty of a comprehensive specification project leads to requirements that are actually neither correct nor comprehensive. Ironically, it seems that almost everybody who has been involved in a public tender projects' specification phase is ready to admit that the produced set of final requirements is not comprehensive, contains contradictory and dependent requirements, is not written in a way that allows iterative and incremental development (i.e., requirements cannot be easily prioritized and split) and does not fully reflect the original vision of the system to be developed.

It also seems that the system requirements—the ones that should specify in a detailed way what the system should do—are often totally forgotten. Instead, an attempt is made to create a system that matches the original system owners' requirements, the ones that never truly were in a form that could be used for verifying whether the original needs were satisfied.

1.4 What We Truly Need

With the requirements specification process explained above, the five main problem areas emerge that are almost inevitably encountered when a project to build the information system starts:

- Problems related to the scope of the developed system.
- Problems related to the volatility of the already specified requirements defining the system.
- Problems related to differences in the expected and the observed quality of the functional and nonfunctional aspects of the system.
- Problems related to the sheer large number of the requirements.
- Problems related to the communication of the true contents of the requirements between the customer representatives and the system provider as well as between the different groups of people, like architects, designers, developers, testers, and management at the system provider side.

When we are aiming at providing an information system as a service—in other words, using the SaaS model—the problems listed above become more prominent than with more traditional system delivery models. With SaaS we are able to provide easily frequent releases for the end-users to use, to easily monitor the user

behavior and to give the users effective feedback channels. Thus with SaaS the problems that otherwise may not have been detected, surface early in the system development phase and need to be addressed already during the early phases of the collaboration with all the stakeholders.

We will briefly discuss the first four problem areas in the remainder of this section. In addition, we will address the fourth one—communication—with a more extensive discussion in Sect. 1.5.

1.4.1 Problems of Scope

Requirement elicitation must begin with an organizational and context analysis to determine the boundary of the target system as well as the objectives of the system. Less ambitious elicitation techniques, not fully addressing this concern, introduce the risk of producing requirements that are incomplete and potentially unusable, because they do not adhere to the users' or organizations' true goals for the system. Performing an organizational and context analysis allows these goals to be captured, and then later used to verify that the requirements are indeed usable and correct.

Elicitation techniques can be overambitious as well. In projects that have inspired this paper, elicitation processes were executed several times during the duration of the system development project and almost every time the goal was set to do proper work this time to find to real requirements. We ended up with a new set of requirements, but it was hard to say if this new set was any better in quality.

Elicitation must focus on the creation of requirements and not design activities in order to adequately address users' concerns. Elicitation strategies which produce requirements in the form of high-level designs run the risk of creating requirements which are ambiguous to the user community. These requirements may not be verifiable by the users because they do not adequately understand the design language. Also, requirements expressed as a design are much more likely to incorporate additional decisions not reflecting user or sponsor needs, in which case the requirements will not be precise or truly necessary for the development.

1.4.2 Problems of Volatility

One of the primary causes of requirements volatility is that user needs evolve over time [19]. The requirements engineering process of eliciting, specifying, and validating the requirements should not therefore be executed only once during system development. We should return to these processes frequently and with low threshold, so that the requirements can reflect the new knowledge gained during specification, validation, and subsequent activities. To this end, any requirements engineering methodology should be iterative in nature, enabling refining existing solutions and creating new ones as knowledge increases [20].

Another cause of requirements volatility is that the requirements are the product of the contributions a number of individuals, with different backgrounds and having most often conflicting needs and goals [21].

Volatility also arises when the users and other customer representatives do not fully understand the capabilities and limitations of the technology and architecture already designed for the system. The lack of understanding leads to unrealistic expectations of either the functionality that can be provided, or the time scale of the development. These expectations should be corrected as early as possible in the requirements elicitation process.

1.4.3 Problems of Observed Quality

When we face huge sets of requirements related to a project or some program aiming for taking an information system into operative use, we rarely seem to verify the comprehensiveness of the requirement set.

The quality of an information system is a complex thing, the quality is experienced by different people in a different way and as the quality consists of a number of diverse aspects, the different people on the customer side observe the quality on totally different basis and also differently at different times.

When defining the quality of an information system, we must understand that there exist at least two important categories of qualities, namely the external qualities and the internal qualities.

External quality. When assessing the external quality, the one that the system users and owners face, at least the following items must be taken into account:

- Observed functional quality of the system.
- Observed usability by real end-users.
- Observed performance of the system, under normal load and under some heavier load during times of congestion.
- Observed reliability of the system in diverse but foreseen conditions.
- Observed maintainability of the system during version upgrades or system patching.
- Observed integrability with the system's true environment.

Above, the word *observed* bears particular significance. One cannot be satisfied with any quality aspect of the system, if the aspect is not validated by measuring or observing it. This requirement makes all the quality aspects above to classify as external quality aspects. The external quality aspects are the ones the acceptance criteria must deal with—and not just some of them, but all.

Internal quality. The quality of an information system is not just external quality observed by an end-user or some other stakeholder. When considering an information system of this scale, the internal quality aspects become as important as the external quality aspects when judging the overall quality of the system in the long run. The internal quality consists of the following aspects:

- Testability. The time of manual testing as a major way to test almost any real-life software system has long gone. Testing needs to be automated to enable functional tests with good coverage and repeatability. With automated tests we can perform hard enough stress testing, long-lasting load testing and even fuzzy testing. Manual testing has its place, though. Nothing can replace a human tester in exploratory testing task.
- Maintainability. A critical system has a long life span. The maintenance of the system, in form of patches, updates, and repairs will form a major expenditure.
- Portability. During the long lifetime the system needs to live through all kinds of environments. Non-portability and lack of robustness will become easily suicidal for the system.
- Supportability. The critical system's configuration needs to be identifiable and we must be able to diagnose behavior and performance and debug it.

Without high quality of these aspects, the pace of the development gradually slows soon down to a halt. In addition, to ensure the high quality of the above-mentioned aspects, one needs to be capable of giving constant attention to the aspects. This again requires a considerable amount of time and energy.

1.4.4 Problems of Expected Quality

With the SaaS paradigm gaining popularity, in addition to the problems related to the observed quality by the users and customers of the provided service, one needs to rethink also how the expectations related to the service have changed because of SaaS.

To be able to benefit from the SaaS, the services need to be designed to be provided as services and delivered in the way SaaS model requires. This means at least that the services fulfill the following requirements:

- Provide easy and maintainable ways for integration with a multitude of customer-specific systems
- Allow customer-specific maintainable configuration
- Support true multi-tenancy
- Provide enough scalability
- Allow regular updates
- Allow fast deployment
- Allow customer-specific upgrade schedules
- Provide world-class security the customers can count on.

Failing to meet these requirements hinders us from capitalizing the benefits made possible by the SaaS model.

Furthermore, as the SaaS model is still for many customers a relatively new thing, we cannot even expect the customer to be able to state these requirements in

any RFQ or alike, as these quality attributes are more or less assumed ones in the era of the SaaS.

1.4.5 Problems of Quantity

There also seems to exist a common problem of confusion with the requirements when the number of the requirements describing the system in question becomes high. When the total number of requirements is in many hundreds or close to thousand, the functional and nonfunctional requirements tend to become mixed up and it begins to be extremely hard to tell apart the nonfunctional part of the requirement from the functional one. Several different requirements may also be expressed together or depend on each other in a way that makes the fulfillment and verification of the single requirement almost impossible.

To the problem of the management of a large number of requirements the available modern requirement management tools provide a partial solution but there still seems to be a human factor that cannot be overlooked: only information systems up to certain size can be understood by single human beings and when the size increases the comprehensive understanding is gradually lost.

1.5 Barriers to Communication

When we have started new information system projects with some totally new or partially unknown group of customer representatives, the first problem faced is always communications. The customer side has typically been involved with the specification process for already some years and in addition to that, has a vast experience on the domain. The system providers, on the other hand, are typically experts in information system creation and not deeply knowledgeable on the specific details or conventions intrinsic to the domain in question. Now, with SaaS all these stakeholders need to understand the customer domain as well, since otherwise providing suitable service level can simply be impossible. Moreover, it is not in clients' interests to continuously educate the staff that runs the service for them—instead, it should be a primary goal of the service provider that they learn domain specifics as rapidly and as effectively as possible.

1.5.1 Barriers at the Customer Interface

A Savant Institute study found that 56% of errors in installed systems were due to poor communication between user and analyst in defining requirements and that these types of errors were the most expensive to correct using up to 82% of

available staff time [22]. Problems of understanding during elicitation can lead to requirements that are ambiguous, incomplete, inconsistent, and even incorrect because they do not address the requirements elicitation stakeholders' true needs. Lack of user input arises when users are not fully aware of their needs or are unable to communicate them. It also arises when analysts and developers fail to ask the necessary questions.

When a system needs to be defined, a series of meeting needs to be held consisting of stakeholders. These stakeholders include clients, users, software engineers, system analysts, domain experts, managers, etc. It has been assumed that having a larger number of people in a meeting helps refining the system requirements and brainstorming becomes much effective and easier. But there is also a potential problem having superfluous and extra stakeholders in a meeting. Furthermore, when we are to deliver possibly tailor-made information system for the customer only once, the meetings make sense and are affordable, but with the SaaS model, arranging constant meetings to define and refine the system requirements may easily become too expensive. Instead, more effective means for getting feedback are required by, e.g., monitoring the service [23].

The language barrier is considered to be a major problem. When there is no proper common protocol to communicate the whole purpose of meeting together is defeated. Different stakeholders may speak literally different languages, e.g., Chinese and English. But even within the same language, it is notorious that stakeholders from different domains (such as management, manufacturing, marketing, and technical) use the same words with different meanings. When literally different languages are used, there is the additional task of translating the relevant documents. When figuratively different "languages" are used, the problem may not even be recognized.

Of course, the lack of clarity in the written down documents also poses a problem if not managed appropriately. Although both the customer and the system provider representatives may have a common understanding on a requirements true content, the precision of the description is difficult to reach without making the requirement document difficult to read.

In the customer interface, according to our experiences, we also seem to play the broken telephone game all too much. The game is played inadvertently when information is passed from customers to consultants or business analysts and only from there to designers, developers and testers.

1.5.2 Internal Barriers

Within the system provider organization, the same barriers to communication exist. Within the development teams we always have some kind of communication problems, being either just lack of discussion or the blind reliance on the information on an issue tracking system's issue instead of asking for better information. The development organization's culture is here in a crucial role: if the culture

encourages people to raise issues and do critical thinking, the barriers to communication can at least partly be overcome.

However, it should be always kept in mind that every time we pass information on, it may get changed and misinterpreted, leading to increased project costs and the delivery of the wrong solutions to our customers and users.

1.5.3 Human Barriers

Wiio's laws [24] are observations about how human communication usually fails, except by accident, formulated in a humoristic way. Since the Wiio's laws point out that the communication always fails, anyone who does understand part of your message will miss the other parts. Consequently, the only way to ensure that the essential information has been communicated is through feedback, which is a necessary part of human communication. However, with feedback we cannot be satisfied if we only get positive and encouraging feedback—only getting positive and negative feedback together indicates that there are people who truly have been trying to understand the communicated issue and are interested enough in it to ensure that what she understood was correct.

An important part of one's reflection is thus also an issue of misunderstanding and its sources [25]. The main point here is that when we are communicating and talking to each other we are rather building a common view of what we are really talking about [25]. Accepting this while handling the requirements of an information system would help us a lot in the process: the starting point should be that the dialogue neither enforces one's opinion against the other's one, nor does it add one's opinion to the other's one—the dialogue will change both of the opinions if the communication is successful.

1.6 Requirements and Architecture

A dual relation exists between requirements and architecture. On one hand, requirements specify what a system must do. On the other hand, architecture describes how a system will be organized and how it will behave in order to fulfill these requirements. As requirements describe the problem and architecture describes the solution, it is easy to think that the requirements naturally precede the architecture. Following this line of thought results in the conclusion that the requirements can be defined without any input from the architecture. However, in any larger scale information system project, all the stakeholders are nowadays ready to admit that the requirements cannot really be specified in detail before the system development starts. This means that the required system architecture, which describes how the system will be organized and determines how it will behave, cannot be appropriately designed. This leaves us with the traditional chicken and

the egg problem: we cannot design the architecture without the requirements and on the other hand, we cannot find the requirements without some form of architecture.

Without the architecture in the picture, high-level external outcomes and constraints often bubble down to use cases, functional specifications, and wireframe UI models. Without architecture, the detailed requirements easily become conflicting and incomplete. Thus the requirement specification and elicitation process without the specification of the architecture does not really make sense. With every set of requirements offered to the potential system providers, there should also be a description of the architecture of the system to be created—and not just some reference architecture, but an architecture the system owners are committed to and which is not be changed easily as requirements evolve.

To generate such architecture—or maybe more realistically a minimum viable architecture (MVA) [26]—a group of architects and domain experts should look for the biggest challenges in the system at hand, especially from the points of view of the deployment environment, the technical issues and the project teams on both sides of the table, the customer and system provider side. The task for this group is to prioritize the identified challenges, find solutions and also debate and lay out alternative solutions. The architects should be able to point out where some approach has potential side effects on other areas and refine the approaches with the domain experts based on the new information.

At the end of the MVA process, the architects and domain experts have collaborated on finding a solution to a problem at hand that would not have been possible to find without the contribution of both the architects and the domain experts. However, while the MVA is not the solution to the entire set of problems encountered during any larger scale information system project, it defines the architecture strategy, the framework and skeleton for the eventual solution.

To make the concept of the MVA a bit more concrete, let us take a look at the steps taken to create a real-life MVA. This example is related to an information system the main purpose of which is to receive, process, refine, relay and store incident-related data[2]:

- We created the most likely scenario of the process the service needs to handle and identify the most critical part of this scenario.
- We analyzed carefully the roles of the users using the service first and their behavior—what kind of users we expect the service to have and without what features these users can still manage?
- We outlined the simplest possible way to achieve an architectural structure, which allows the service to provide the users the functionalities needed for the most likely scenario, the MVA.
- We made sure we understand all the nonfunctional requirements related to this most typical scenario, like performance, scalability, security, reliability and integrability related requirements. For the security and the performance

[2]The MVA is taken out of its context to protect the identity of the system in question.

requirements a good and realistic understanding of the number of users and the amount of expected data is needed—no guesses.
- We wrote down a list of what identified functional and nonfunctional requirements cannot be met with the architecture outlined—all these related to other than the most likely usage scenario. We tried to identify the hardest potential problem areas in the MVA but did not try to solve them in this early phase of the development process.
- We accepted the fact that the MVA will not be the final architecture, but it will be good enough to start with and to get the development process moving on.

1.7 Guidelines for the Transformation

Based on our experiences, we do not assume that any quick changes will happen in the fashion private and public sector information systems will be procured. Instead, we acknowledge that we must deal with the IEEE 830 styled requirements, or "questions" as the requirements are sometimes disguised as, in the foreseeable future. Accepting this, we next start to look for ways to reduce the damage or at least the harm the suboptimal way of expressing the system owner needs the requirements typically cause.

Insisting to know why. We need to understand and accept the fact that the requirements are just sentences written by some people at some point of time with some level of understanding of the situation. If the requirements do not experience transformations and we do not dispose or create new ones during the system development project, we are not responding to the actual needs of the system owner. Especially, the nonfunctional requirements may be very difficult to state precisely. On the other hand, imprecise requirements are not of use, they easily become expensive to implement and are almost impossible to verify reliably. Some simple techniques, like the Five Whys [27], could be employed in the discussion with the system owner's to promote deeper thinking through questioning and is easily adaptable to most problems.

Maintaining agreed practices. No matter how hard the customer and the system or service provider both wish to execute a program or project in an agile fashion, joint practices and processes must be carefully agreed upon. Agility does not mean that we should invent the ways of working every day. Any larger information system project needs to have fully agreed and also documented ways of working related to all interaction between the customer and the system provider. This does not mean naturally that the processes should be inflexible and bureaucratic—they just need to be agreed, known and followed by all the stakeholders in a disciplined way.

Listening the right way. One of the buzzwords in the information system domain has been for long already that the system provider should be customer driven. This means that the system providers should always be aware and respond to customer problems swiftly. However, this approach can also be fatal when the

communicated requests from the customer side at different points in time have varying and differing views of how the system should be developed. Cooper [28] stresses the difference between listening to and following a customer. While listening to a customer is good, following a customer by merely doing what a customer tells you to do is not. The customer that the system provider hears best—"the loudest voice"—may not be the most important customer: it may even be that we should not even have this customer!

Importance of continuous interaction. The relationship between the customer and the system provider may not always be easy and all kinds of adversities will be encountered during a long-lasting project. Nevertheless, despite the situation, the communication channels should always be open and utilized on a daily basis. In our experience, no information system project has failed due to excessive communication. If nothing else, the customer representatives should be regularly contacted by the system provider representatives to ask them how they are and how they feel about the project at hand. It should also be kept in mind that a feature management system or a bug or issue tracking system is never a replacement for interactions with end-users and other stakeholders at the customer side. The actuality of human communication lies in the fact that the communication neither enforces one's opinion against the other's one, nor does it add one's opinion to the other's one. The result of the communication is that both of them are changed [25].

Identifying the most valuable features. One of the downsides the continuous interaction with the customers and the end-users has it that along with wideband and frequent communication we inevitably also talk about features and qualities that the customer representatives or the end-users think they would like to have, the so called nice-to-have features and qualities. New features are extremely easy to invent and fun to discuss about. However, each feature should be associated with explicit stakeholder value and prioritized accordingly.

Strict no to feature creep. Since the information system projects are nowadays typically executed in a close co-operation with the end-users and other customer-side stakeholders, the system provider receives easily a lot of feedback on the functional quality aspects of the system under development. When receiving the feedback, the system provider should be very careful to avoid the feature creep [29]. However, the responsibility on keeping the scope set by the system owner should not be on the development teams solely but on the customer representatives and end-users. It seems to be somehow many times forgotten that keeping the feature creep under control is of the highest interest for all stakeholders. In case feature creep takes place, the whole project is immediately in danger and in the worst case no system will be ever created.

To avoid feature creep, one needs to perform rigorous and visible change management. Learning to say "No" in a nice way to customers and end-users is obligatory.

1.8 Conclusions

Although a considerable amount of work has been put into creating and refining the requirements for all kinds of information systems, there still seems to be a lot to do in order to make the development processes produce what is truly needed in an effective way. We have problems with the scope, volatility and quality of the requirements and in particular, problems with the communication related to the requirements. The problems result in misunderstandings, a lot of confusion and eventually waste of time and resources.

To overcome the requirement related problems in system development efforts, no silver bullet exists. However, critical thinking and willingness to truly understand the customer, strict prioritization and disciplined ways of working with the requirements and wideband frequent communication help a lot.

In our opinion, system owners and stakeholders are not to blame for the shortcomings that can be traced back to bad or missing requirements, encountered during the design and development of an information system. The people who are experts in the system design, development, and deployment are responsible for educating the system owners and all other stakeholders regarding how to identify and express the true needs in a form that enables more error-free communication and full comprehension of the issues involved on both sides.

In addition, starting from quality needs instead of functional requirements gives us better chances to find the right solutions to the true needs. One crucial step toward being able to specify the nonfunctional requirements would be the usage of minimum viable architecture (MVA), as an intrinsic part amending the functional and other customer requirements. Doing such architecture requires a lot of trust between the service providers and the system owners, but motivation for this change should be clear: with better quality of requirements and better communication we would be seeing better new information systems, longer information system life span, and eventually happier customers.

References

1. Leffingwell, D., & Widrig, D. (2000). Managing software requirements: a unified approach. Addison-Wesley Professional.
2. Stakeholder Needs and Requirements. (2015, December 18). in BKCASE Editorial Board, Guide to the Systems Engineering Body of Knowledge (SEBoK), version 1.5.1, R.D. Ad-cock (EIC), Hoboken, NJ: The Trustees of the Stevens Institute of Technology ©2015. Retrieved 14 Mar 2016 from http://sebokwiki.org/w/index.php?title=Stakeholder_Needs_and_Requirements&oldid=51430.
3. Hochstetter, J., & Cares, C. (2012, November). Call for Software Tenders: Features and Research Problems. In Proceedings of the 7th International Conference on Software Engineering Advances (ICSEA (Vol. 12).
4. Jurison, J. (1999). Software project management: the manager's view. Communications of the AIS, 2(3es), 2.

5. Marilly, E., Martinot, O., Betgé-Brezetz, S., & Delègue, G. (2002). Requirements for service level agreement management. In IP Operations and Management, 2002 IEEE Workshop on (pp. 57–62). IEEE.
6. Hofmann, H. F., & Lehner, F. (2001). Requirements engineering as a success factor in software projects. IEEE software, 18(4), 58.
7. Jones, C. (2008). Applied software measurement: global analysis of productivity and quality. McGraw-Hill Education Group.
8. Munassar, N. M. A., & Govardhan, A. (2010). A comparison between five models of software engineering. IJCSI, 5, 95–101.
9. Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: Review and analysis.
10. Ralph, P. (2013). The illusion of requirements in software development. Requirements Engineering, 18(3), 293–296.
11. Davis, A. M. (1993). Software requirements: objects, functions, and states. Prentice-Hall, Inc.
12. Wikipedia contributors. "Request for proposal." Wikipedia, The Free Encyclopedia. 7 Feb. 2016. Web. 14 Mar. 2016.
13. Wikipedia contributors. "Request for quotation." Wikipedia, The Free Encyclopedia. 2 Feb. 2016. Web. 14 Mar. 2016.
14. Wikipedia contributors. "Request for information." Wikipedia, The Free Encyclopedia. 20 Feb. 2016. Web. 14 Mar. 2016.
15. IEEE Std 830-1998 (Revision of IEEE Std 830-1993), IEEE Recommended Practice for Software Requirements Specifications.
16. Chen, L., Babar, M. A., & Nuseibeh, B. (2013). Characterizing architecturally significant requirements. IEEE software, 30(2), 38–45.
17. Frakes, W. B., & Kang, K. (2005). Software reuse research: Status and future. IEEE transactions on Software Engineering, (7), 529–536.
18. Kramer, J., & Magee, J. (2007, May). Self-managed systems: an architectural challenge. In Future of Software Engineering, 2007. FOSE'07 (pp. 259–268). IEEE.
19. Brooks, FP Jr. "No Silver Bullet Essence and Accidents of Software Engineering." Computer 4 (1987): 10–19.
20. Christel, Michael G. and Kang, Kyo C. "Issues in Requirements Elicitation", CMU/SEI-92-TR-12, 1992.
21. Ebenau, B. and Strauss, S., Software Inspection Process. McGraw-Hill, 1994.
22. Goodrich, V., & Olfman, L. (1990, January). An experimental evaluation of task and methodology variables for requirements definition phase success. In System Sciences, 1990. Proceedings of the Twenty-Third Annual Hawaii International Conference on (Vol. 4, pp. 201–209). IEEE.
23. Koski, A., Kuusinen, K., Suonsyrjä, S., & Mikkonen, T. (2016). Implementing Continuous Customer Care: First-hand Experiences from an Industrial Setting. In Proceedings of the 42nd Euromicro Conference on SEAA.
24. Wiio, O. A. (1978). Wiion lait—ja vähän muidenkin (Wiio's laws—and some others'). Weilin +Göös.
25. Klimova, B. F., & Semradova, I. (2012). Barriers to communication. Procedia-Social and Behavioral Sciences, 31, 207–211.
26. Erder, M., & Pureur, P. (2015). Continuous Architecture: Sustainable Architecture in an Agile and Cloud-centric World. Morgan Kaufmann.
27. Serrat, Olivier. "The five whys technique." (2009).
28. Cooper, A. (1999). The inmates are running the asylum: [Why high-tech products drive us crazy and how to restore the sanity] (Vol. 261). Indianapolis: Sams.
29. Elliott, B. (2007, July). Anything is possible: Managing feature creep in an innovation rich environment. In Engineering Management Conference, 2007 IEEE International (pp. 304–307). IEEE.