Acknowledgement

This thesis is to mark the completion of my Master degree program at the Department of Informatics, University of Oslo. Throughout the process, I have engaged myself to the research topic as much as the educational aspect of the work. Yet, I have been looking forward to this moment, when those seemingly never-ending days of reading and writing finally come to an end.

This work would not be possible without the kind support of my supervisor, Sundeep Sahay, from whom I learn a great deal about the topic, the research and writing process, and more subtly about the life of a researcher.

My gratitude to Mr. Venkata Rao Mallineni and the staff at IndiSoft who have been so kind and supportive during my fieldwork. Thanks also to my good friends in Padmarao Nagar, with whom I had a great time during my stay in Hyderabad, India.

I have enjoyed a constant moral support from my father Syahrir Ibrahim, my mother Ratna Syahrir, and my two sisters Irwina and Iryani. To my uncle Tarmizi Taher and my aunt Djoesma Tarmizi, thank you for everything, especially for opening the way.

Life as a student in the cold Norway has been warmed by my dear friends from different countries, to name a few: Sudan, Ethiopia, India, Norway and Indonesia. To Mohammed Sidahmed, thanks for listening. To Sherly Saragih, thanks for always being there when I need it most.

Kringsjå, 1 November 2005

Contents

List of Figures

Chapter 1

Introduction

"The bitterness of poor quality remains long after the sweetness of meeting the schedule has been forgotten (anonymous)"

This chapter presents the outline of the study which is aimed to understand issues of quality in software development processes in the global software outsourcing (GSO) context. In section 1.1, the significance of GSO is presented. In section 1.2, contemporary views of software quality are discussed, started with defining the meaning of quality in general followed by discussing the current approaches to software quality. In section 1.3, quality related challenges in GSO are discussed. The objectives of this research are presented in section 1.4. Finally, the structure of the thesis is introduced in section 1.5.

1.1 The Significance of GSO

Global outsourcing is one of the fastest growing and evolving business activities worldwide. Following the outsourcing of Eastman Kodak's IT operations in 1989, IT outsourcing in the US alone grew to $101bn by 2000 and has been predicted to reach $160bn by 2005. Global demand has enabled several developing nations to transform key sectors of their economies – something many other countries aspire to emulate. Skill shortages and the emergence of new low cost providers continue to fuel the growth of outsourcing, while developments in information and communication technologies (ICT) are extending its scope [Sloper, 2004].

Currently, there are a large number of development methodologies and approaches available to software practitioners. Yet, defective software is common and high quality software is scarce. Many software projects con-

tinue to fail, despite the promising methodologies or approaches that they use. Quality in software development is still an unresolved issue.

Lessons have been learnt from many failed software projects. For example, Sierra – a UK-based software house – closed their Bangalore operation, which had been set up with a high level of optimism and expectation, and which failed to address challenges related to global outsourcing such as distance in all its connotations – geographical, cultural, and linguistic [Heeks et al., 2001]. A Russian-Norwegian software project had to struggle out of a difficult period by overcoming the problematic areas around knowledge management, especially related to domain knowledge, language and project management [Imsland and Sahay, 2005].

Today, IT organizations can outsource two basic types of work: explicit functions relevant to the operations of IT (for example, software development and infrastructure), and business operations that have a direct impact on IT systems (for example, customer call centers and manufacturing) [Laplante et al., 2004]. The first type, global software development, also referred to as global software work, is the main focus of this study, with an emphasis on issues relating to developing and assessing the quality of software.

Many people agree that quality has been an issue in software development for years. A recent report states that after five decades of software development, defective software is the norm, high quality software the exception [Malloy and Voas, 2004]. There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies [Jackson, 2004].

Most studies and methodologies to determine quality of software are related to those built in co-located settings. Little is known about these issues of quality in the context of software developed in globally distributed contexts.

The thesis thus presents a study aimed at analysing the determinants of software quality in global software development, and the ongoing challenges and approaches to achieve it. Empirical research has been performed in a software company in India which develops software solutions for the worldwide market. Two projects have been selected as case studies, which provide the empirical basis for the analysis. In the following section, we will examine the contemporary views of software quality.

1.2 Contemporary Views of Software Quality

This section briefly summarizes the current views of software quality, starting with a general discussion on quality, followed by more specific description of some widely accepted approaches to quality.

Quality has always been the aim of any human endeavour whether it is deliberately declared or implicitly taken for granted. An interesting example is pointed out by Pyzdek (2003) that quality issues have been a matter of great concern for people throughout recorded history. Quality issues have been written in law since the Mesopotamian era. The Babylonian king Hammurabi (1792-1750 BC) made Babylon the chief Mesopotamian kingdom and codified the laws of Mesopotamia and Sumeria. The Code of Hammurabi called for the death of builders whose building collapsed and killed its occupants. No doubt the builders of the time took great care to construct high quality structures! [Pyzdek, 2003]

Engineering or manufacturing is one of the human enterprises which are particularly concerned with quality issues. Measures are taken to control every stage of the production system. They seem to follow the simple formula which is stated as "with good raw materials as inputs, well-calculated processes and accurate tools, quality of the products will be assured". For most cases, that formula works well in traditional manufacturing. Software development, which till not long ago was perceived as a pure engineering enterprise, unfortunately does not share the same prescription. Software development differs from traditional manufacturing due to its unique characteristics imposed by the character of its product: the software; and its processes such as the underlying development methodology.

Software is a logical system element instead of physical; therefore it implies the closeness between the creation (software) and the creator (programmer). Human and social elements play more influential roles in the production of software than in hardware. Human tendency to produce error in their actions is reflected in software development and magnified in larger projects involving more people, which eventually leads to the phenomenon coined as "software crisis". Software crisis, observed for the first time in the 60's, refers to the inability to develop software on time, on budget and within requirement([Brooks, 1987], [Feller and Fitzgerald, 2000], [Bryant, 2000]). In this thesis, software crisis is perceived as the failure to achieve the desired quality of the software product.

Software engineering was, at that time, the answer to "the software crisis". The development of systematic procedures to produce structured code, which became known as 'methodology', was the first widespread

attempt to take account of quality issues during software development. However, this creation of methodologies still could not wipe out the software crisis completely. The crisis was, and is, still there.

This thesis seeks to address the concerns of such a software crisis with regard to the quality ideas in the context of global software outsourcing – one of the current modes of working which poses new challenges to software development. Before continuing that, in the next section, I would like to briefly discuss the different perspectives on software and its quality. We need to know more about what quality means to us before investigating this matter in different contexts of software development.

1.2.1 What is Quality?

Before starting a discussion on quality, it would be practical to have a well-established definition of it at our disposal. Unfortunately, quality is hard to define, impossible to measure, but easy to recognize. It is generally transparent when present, but easily recognized in its absence, for example when a piece of software fails to perform properly.

Quality is not absolute; it means different things in different situations. Quality is multidimensional; it has many contributing factors which are not easily summarized in a simple, quantitative way as some aspects of it can be measured while some others may not. Quality is subject to constraints; assessment of it in most cases cannot be separated from cost or other critical resources such as people, tools, and time. Quality is about acceptable compromises; where quality is constrained and compromises are required, some quality criteria may be sacrificed more acceptably than others. Further, quality criteria are not independent, but interact with each other, often causing conflicts [Gillies, 1997].

The first principle of modern quality improvement is that "the customer is king!" Relatively-perceived quality is not the same thing as traditional "conformance" quality. Many a company has struggled to obtain a high conformance rate, only to be disappointed with the results in the marketplace. What matters is what your customers think about your quality [Pyzdek, 2003].

Many people have attempted to define quality, which tend to be mostly context-dependent, such as: "the degree of excellence" (Oxford English Dictionary, 1990); "zero defect" [Crosby, 1979]; "the totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs" (ISO 1986); or the pragmatic one, "quality is when the customer comes back, not the product" [Frühauf, 1994]. Sommerville

(2004) argues that good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable [Sommerville, 2004]. Kitchenham (1989) refers to software quality as 'the fitness for needs' and claims quality involves matching expectations. The definition recognizes two questions of quality software: (1) is it a good solution? (2) does it address the right problem? These two questions are a good starting point for us to delve into the quality concept in software [Kitchenham, 1989]. In the following section, current approaches to software quality are examined.

1.2.2 Contemporary Approaches to Software Quality

In this subsection, a more specific discussion on quality in software development is presented. Most software quality factors are directly measurable only after the software system has been deployed [Khoshgoftaar et al., 2004]. This is one of the reasons why software quality has to be approached in a different manner than, say, traditional manufacturing.

Since quality happens to be difficult to define, I choose to learn from how its development has been approached. In this section, two strands in the development of ideas around quality and approaches within software development will be examined: software engineering and quality management system. Quality approach based upon software engineering seeks to apply rigorous engineering practices to software development and the alternative is the application of quality management ideas [Gillies, 1997].

As briefly mentioned earlier, software engineering was introduced in the 60s – when the term 'software crisis' was first coined – to try to formalize the development of software, using ideas from other engineering disciplines. Software engineering is the processes, methods, and tools that support a software development team in building a high-quality and useful software system [Filman et al., 2004].

In general, most people view software engineering largely as a deterministic technical enterprise with some human and social adjuncts, collectively referred to as human factors. And, while our community is constantly redefining and expanding the notion of human factors in software engineering, a broader understanding of how the human and social environment affects software engineering escapes us [Sharp et al., 2000].

Software Development Life Cycle (SDLC, also known as the waterfall model, was, and possibly still is, the mainstream methodology in software engineering. The processes prescribed by the methodology include require-

ment specifications, design, development, and testing. There are also practices which support the aforementioned ones throughout its application of those processes in a software project, i.e. software documentation. Several alternatives to this mainstream approach have been developed such as, to name a few, the prototyping, evolutionary model, object orientation and agile development. Within the domain of software engineering, quality continues to remain an issue of concern. Although remedies such as fourth generation programming languages, structured techniques and object-oriented technology have been promoted, a "silver bullet" has yet to be found [Ward and Aurum, 2004].

Evolutionary process models were introduced because software, like all complex systems, evolves over a period of time. Business and product requirements often change as development proceeds, making a straight-line path to an end product unrealistic. In these situations, software engineers need a process model that has been explicitly designed to accommodate a product that evolves over time. Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software. The intent of evolutionary models is to develop high-quality software in an iterative and incremental manner [Pressman, 2005] p.83.

Often, a customer defines a set of general objectives for software, but does not identify detailed input, processing, or output requirements. In such cases, and many other situations, the *prototyping* paradigm may offer the best approach. Although prototyping can be used as a standalone process model, it is more commonly used as a technique that can be implemented within the context of other process models. It mainly assists the software engineers and the customer to better understand what is to be built when requirements are fuzzy [Pressman, 2005] p.83.

Some concerns about the evolutionary models, including prototyping, have been expressed. First, they pose a problem to project planning because of the uncertain number of cycles required to construct the product. Second, those models do not establish the maximum speed of the evolution. Third, software processes should be focused on flexibility, extensibility, and speed of development rather than on high quality [Pressman, 2005] p.90. In my opinion, this evolutionary approach does address the process and product dimensions of software by embracing iterative development to reach high quality product, but it does *not* explicitly give much credit for the role of people in the process, whereas people are assumed to act accordingly to the prescribed steps in each iteration.

In early 2001, Agile Software Development was introduced by its supporters with a promising claim of uncovering better ways of develop-

ing software. They presented their values formulated as the following: individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to changes over following a plan. Various lightweight methodologies which belong to this movement are: Adaptive Software Development, eXtreme Programming (XP), Scrum, Crystal, Feature-Driven Development, Dynamic System Development Method (DSDM), and "pragmatic programming" (Cockburn 2001). One of the distinguishing characteristics of this methodology is its emphasis on *the role of people* in software development. Despite this new and relatively fresh perspective on software development, Agile Development specifically mentions its limitation concerning its use in virtual teams in multi site development, offshore development and distributed development. Virtual is a euphemism to mean not sitting together.

Another approach to quality in software is the use of Quality Management System at the organizational level. Quality Management System – herewith referred by its acronym, QMS – is the organizational structure, responsibilities, procedures, processes and resources for implementing quality management (ISO8042, 1986). QMS goes further than a methodology in ensuring that responsibility is clearly established for the prescribed procedures and processes. If the methodology is intended to lay down which procedures should be carried out, QMS should ensure that the procedures are actually carried out to the required standard. At best, it provides a disciplined and systematic framework; while at worst, it can become a bureaucratic nightmare. Several standards are available for quality management and process improvement such as ISO9000 and CMM.

Gillies (1997) states that the main concern about the model of a quality system which forms the heart of ISO9001 is its emphasis on quality control procedures and there is very little in it about establishing *a human quality culture*. Without the establishment of a quality culture and a formal requirement for procedures to facilitate the process, the vital process of continuous improvement which takes quality management beyond the recording of errors and performance may be omitted. Another common complaint about ISO9001 is that it tends to fossilize procedures – making them outmoded, rigid, or fixed – rather than encourage process improvement.

Despite the problems of ISO standards, a range of standards and models have been developed which seek to provide the benefits of quality standards whilst recognizing different stages of development and the continuous need to improve. Prominent among these efforts is the SEI's Ca-

pability Maturity Model (CMM). The CMM, developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in Pittsburgh, is an approach for evaluating and measuring the maturity of the software development process of organizations on a scale of 1 to 5. The purpose of CMM Integration is to provide guidance for improving your organizations processes and your ability to manage the development, acquisition, and maintenance of products or services. It has been used extensively for avionics software and for government projects since it was created in the mid-1980s. The Software Engineering Institute has subsequently released a revised version known as the Capability Maturity Model Integration (CMMI). Similar to the line of thought mentioned earlier in this section, the basic premise underlying this model is that the quality of a software product is largely determined by the quality of the software development and maintenance processes used to build it. A maturity model is a method for judging whether how and why processes are used, are characteristic of a mature organization. Stage models offer insights into how computer-based IT and managerial and organizational strategies evolve and mature over time. According to stage models, organizations progress through a number of successive, identifiable stages. Each stage reflects a particular level of maturity in terms of the use and management of IT in the organization [Fairchild, 2004].

This maturity model is defined as a five-level framework for how an organization matures its software processes from ad hoc, chaotic to mature, disciplined ones. The software process is defined as a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products. The model assumes that as an organization matures, the software process becomes better defined and more consistently implemented throughout the organization. CMM, and other similar models such as Software Process Improvement and Capability dEtermination (SPICE), etc, do not exclude ISO, but they do provide a mechanism to improve the existing ISO practices. They allow an organization to build an evolutionary path to the level of quality that they wish to achieve.

To conclude, there are multiple views of software quality, and defining quality is hard, measuring quality is impossible, but recognizing the presence of quality is easy – even more so with its absence. Therefore it is more practical to examine the approaches to produce and implement quality. The current view of software quality is marked by the perception of software as a product resulting from careful processes which are conducted by people. This perception will be discussed further in the following chapter as the three dimensions of software. Concrete attempts to achieve quality

have been undertaken through methodologies and standard quality management systems which are based on one or more of those perceptions. However, while the current challenge of the software crisis is not yet completely solved, our understanding of these issues in global software context is very immature. I now discuss some of these issues in the following section.

1.3 Quality Related Challenges in GSO

This section discusses the quality related challenges in global software outsourcing context. The context itself is briefly presented and then the aforementioned approaches to software quality are re-examined with respect to the contextual challenges.

Global Software Outsourcing is the offshore development of software by personnel outside the client's home country [Barret et al., 1997]. A naïve viewpoint may suggest that there is nothing new about software development in a global outsourcing context. Software would always be software regardless of the context and therefore all the methodologies to develop it would always be applicable anytime and anywhere. The use of the Information and Communication Technologies (ICT) to transfer all the necessary material for the job – even the work itself – and the application of one or more appropriate methodologies would, magically, produce quality results! A further argument is that people have been doing outsourcing in traditional manufacturing such as automobile or other goods anyway. If one can produce quality software in one place, why not reapply the exact same way in another? Unfortunately, in reality it is not that simple.

Recent movements in software methodologies, such as Agile Development, have identified the importance of people's role and communication processes in determining the success of software development. The reason that many applications based upon traditional methodologies fail is partly because those methodologies do not put people at the centre as one of the determining roles, who actively conduct the process and produce results bearing all their human psychological and social aspects. Methodologies like SDLC take for granted that people will act uniformly to the prescribed actions defined in the methodologies. Agile software development emphasizes communication as an important aspect of the process as in outsourcing. There are a variety of people involved in the work setting, and without rich communication due to time and distance constraints. The following section will briefly examine the outsourcing context more closely, followed by the elaboration of the challenges in applying the two quality

approaches presented before, namely Software Engineering and QMS.

1.3.1 Outsourcing Context

In this section, the context in which GSO takes place is presented. One key domain of change characterized by globalization processes at the turn of the twenty-first century is in the international business environment and organizational forms which are being reshaped as part of a new scenario that have diversely labelled as the 'new economy', 'digital economy', 'network society' or the 'information age'. The new economy is informational, global, and networked – the characterization which refers to its fundamental distinctive features. It is *informational* because the productivity and competitiveness of units or agents in this economy (be it firms, regions, or nations) fundamentally depends upon their capacity to generate, process, and apply efficiently knowledge-based information. It is *global* because the core activities of production, consumption, and circulation, as well as their components (capital, labour, raw materials, management, information, technology, markets) are organized on a global scale, either directly or through a network of linkages between economic agents. It is *networked* because, under the new historical conditions, productivity is generated through and competition is played out in a global network of interaction between business networks [Castells, 2000].

Today's global information economy strongly encourages forms of development that bring together participants from across geographical locations, time zones, and business organizations [Yilmaz et al., 2004].

The new economy has enabled a new organizational form which enables the conduct of Global Software Work (GSW) to emerge. GSW refers to software work undertaken at geographically separated locations across national boundaries in a coordinated fashion involving real-time or asynchronous interaction [Sahay et al., 2003]. GSW can thus include work done across global borders through, for example, outsourcing and work in a global team within a multinational company.

GSW takes place within an extremely dynamic and diverse global marketplace that is populated by organizations, big and small, from countries both developed and developing. Diversity, complexity and uniqueness are inherent to GSW making them an exciting and relatively unexplored domain of study [Sahay, 2003]. Sahay (2003) states that GSW is based on the assumption that software projects can be subdivided into relatively independent and autonomous modules, and pieces of work can be distributed and coordinated through the use of ICTs across the globe. The

modularization of work, its distribution across different development centres and subsequent integration requires standardization of various products, processes and also practices. Furthermore, the black-box approach – that is, the use of formal project requirements to transfer knowledge about the application problem domain from the client to the vendor organization – has long been the mainstay of outsourced software development [Tiwana, 2004]. In the next section, we examine the challenges in GSW, starting with the application of software methodology.

1.3.2 Software Methodology Challenges in GSW

Applying a certain methodology slavishly does not ensure success in achieving quality, not to mention that no methodologies are able to guarantee their successful application in all problem situations. The success or failure of development efforts cannot be attributed exclusively to the use, misuse, or nonuse of methodologies [Avison and Fitzgerald, 2003b]. In the GSW context, this fact is magnified by its inherent constraints of *time*, *space* and *culture* separation. Those constraints, again, reflect the importance of people since they affect directly the people aspect of software development – such as collaborative aspects in communication, as opposed to simply exchanging information – more than the product or process aspects.

There has been a worldwide movement towards process improvement and many different paradigms have been created that can potentially benefit today's software projects. More people realize that the success of software projects and its quality management systems with an emphasis on the process improvement depends, to a greater degree, on social factors. The difficulties of achieving social acceptance for Software QMS will also be exacerbated in the future by the increased globalization of the software market and the use of cross-cultural development teams within multinational companies. An integration of a disciplined approach, focusing on repeatable processes and continuous improvements, together with an emphasis on people and culture within the organizational context will add value to the software industry [Siakas, 2002].

As we know, human, social and cultural factors in software engineering are crucial factors. Software engineering's very nature is mutually shaped by the human and social world in which it exists [Sharp et al., 2000]. Sharp et al (2000) also states that a distinct culture of software engineering transcends national, regional, and organizational cultures.

However, the latest trend of agile software development, which recognizes and emphasizes the importance of people and communication,

specifically states that the application of agile methodologies requires co-location as the primary condition. For example, Cockburn (2001) states: co-location is considered a critical element in Crystal Clear. Crystal Clear is one part of Crystal methodology family which he proposed. Crystal Clear is a *light* methodology for small teams (1-6 people). The weight of a methodology is the product of its "size" (the number of control elements, e.g. deliverables, standard, activity, quality measures, and techniques description) and "ceremony" (the amount of precision and the tightness of tolerance in the methodology). A rule of Crystal Clear is that the entire team must sit in the same or adjacent rooms, in order to take advantage of the "convection currents" of information and the "osmotic communication" – analogies borrowed from fluid theory denoting the fluidness of information [Cockburn, 2001]. Co-location is not possible in GSW, which by its very nature is distributed, thus implying that such methodologies are not applicable to the context.

Thus, methodologically speaking, there are two challenges in the GSW context: (1) traditional methodologies, which emphasize control on mainly the processes and products, are considered inadequate and problematic, especially concerning the people aspect; (2) agile methodology is simply impossible to apply due to its conditional requirements of co-location. From the academic point of view, those challenges present an interesting area to study because they raise many questions such as: how far can traditional methodologies still help with the outsourcing project? One may also speculate with the idea of creating another approach or methodology for GSW or merely finding a certain way to tailor the available methodologies to suit the context. In the following section, another approach to quality, i.e. standards and QMS, is presented.

1.3.3 Standards and QMS in GSW

In this thesis, the use of the terms standard and QMS are closely related and interchangeable. When I use the term "standard" I also mean "quality standard" because QMS in itself is a form of quality standard. Researchers have shown that imposing standards in GSW is not an easy task. Sahay (2003) has argued that GSW is based on the assumption that software projects can be subdivided into relatively independent and autonomous modules, and pieces of work can be distributed and coordinated through the use of ICTs across the globe. The modularization of work, its distribution across different development centres and subsequent integration requires standardization of various products (for example, the software de-

velopment environment used), processes (such as software development methodologies) and also practices (such as reporting routines). In the context of GSW, standards are conceptualized as a process of simplification and abstraction with the aim to define and communicate significant aspects of the processes, artefacts and structures across time and space. The aim is to enable some form of universalisation and mass production. This process of standardization is extremely complex as it involves questions of what and how much to standardize, so as to best develop a pragmatic balance between the need for universal templates with the demands of being sensitive to local particularities.

In a global outsourcing relationship, a constant attempt is to build and sustain a relationship by homogenizing operations to the extent that the outsourcing and outsourced firms cannot be distinguished from each other. For example, firms try to simplify and coordinate tasks by standardizing various processes of knowledge transfer, such as how project reports are written, and the criterion to judge the *quality* of a developer's work. These standardized systems, often codified in manuals and databases, and sometimes implicit and unwritten, serve as points of reference to coordinate work across time and space. GSW reflects characteristics of other forms of global work in general where the focus is on developing standardization, productivity, and efficiency. It involves the application of various kinds of knowledge systems including programming languages, software development methodologies, project management techniques, and the application domain. Different programming languages are used in software development for both general purpose and specialized domains [Sahay, 2003].

The quest for standardization reflects and is inscribed in the Software Engineering tradition. Both standardization and software engineering attempts to impart structure and predictability into processes to minimize the heterogeneity of software work, for example, with the standardization of development processes, methodologies and programming languages. Furthermore, standards in GSW relationships are conceptualized to be largely negotiated 'internally' at the social, political, and cultural levels by the involved parties. The interest in standards extends beyond the technical concerns of individual systems or the protocols to include the relationship in its totality, the standards for technical and physical artefacts, software development processes, and other formal or informal management practices. Again, this is because in software development, quality of the software product cannot be decided until late in the development cycle, thus the reliance has to be placed on process quality as an expected means for achieving product quality.

Standardisation in GSW is challenging because it is like shooting a moving target. Sahay (2003) emphasizes the extremely dynamic nature of standards and presents an analysis of some of the mechanisms through which these changes take place. These mechanisms are shaped by individual actions, organizational policies, industry wide changes, and changing expectations within a processual relationship. He shows the limits of what can be standardized and how much. From the perspective of quality, this insight can be related to the decisions to apply the available quality standards, such as ISO9000 series and CMM, to the GSW context. Practicing managers are expected to know how much those standards may contribute to the overall quality goals and to take necessary actions to fulfil the gap.

This discussion adds one more question to answer in the pursuit of quality: if possible at all, how effective can ISO, CMM or other QMS related approaches assure the quality achievement in GSW? In addressing this, it will be useful to examine the software practices used in India, a country considered a leader in GSW and also in having companies with a high level of quality certification.

To conclude, this section has discussed some of the challenges in applying standard approaches to quality in the GSW context. GSW poses some inherent constraints including, time differences, geographical separation and cultural differences. These constraints have been the source of challenges to the application of the present dominant approaches to quality: software engineering and QMS. In the following section, the research objectives are introduced.

1.4 Objectives

Within the problem area described above, this thesis aims at the following:

> ”**Identify and describe software quality determinants in global software work.**”

Field research was conducted in an Indian-based software company providing for worldwide market. Data from two projects, which provide the empirical basis of the analysis, have been collected along with the interview and observation notes. The next section presents the structure of the thesis.

1.5 Structure of the thesis

This chapter has provided the motivation and outline of this study. Chapter 2 presents the literature review and the proposed quality framework. The research methodology and empirical setting are described in chapter 3. The organization, as a context in which the two project case-studies took place, is discussed in chapter 4. Chapter 5 and 6 are dedicated to each case study, in which the project is described and the quality related activities are identified. Chapter 7 presents the discussion and chapter 8 concludes the thesis.

Chapter 2

Literature Review and Proposed Quality Framework

This chapter presents a review of relevant literature and proposes a framework to analyse quality practices. It begins with a discussion on software and its dimensions in section 2.1. Section 2.2 discusses some quality measurement efforts. In section 2.3, Software Engineering methodologies as an attempt to achieve quality are examined; some widely known methodolgies are presented including several common software practices across those methodologies. Section 2.4 presents QMS, its elements and some well-known examples. Finally, in section 2.5, a quality framework is proposed for further use in the case-study analysis.

2.1 Software and Its Dimensions

This section introduces software as a multidimensional entity. In this thesis I delimit the scope of software here as the programming system product as defined by [Brooks, 1995] as follows:

> "a collection of interacting programs, coordinated in function and discipline in format so that the assemblage constitutes an entire facility for large tasks, which can be run, tested and extended by anybody (not only its author). It should be usable in many operating environment. It is written in a generalized fashion. It has been tested with a substantial bank of test case including extensive testing with other system components in all expected combinations and the result of the test should be recorded. It should come with documentation. There is conformity of inputs and outputs with syntax and semantics with

> precisely defined interfaces. Its use of resource is well calculated. (p.6)"

This definition is somewhat exhaustive and covers some of the practices in software engineering such as testing, configuration and documentation. However, this definition does not really show why software development is different from traditional manufacturing or engineering which is defined as "the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people" (Merriam-Webster's Collegiate Dictionary, 1997).

Software is vastly different from other things that human beings build: it is primarily a logical rather than a physical system element. The character of software development sometimes seems closer to mathematics and art than most other engineering disciplines. Software is inherently an intangible, intellectual development medium. No laws of physics govern its behavior; it is both marvelously and dangerously malleable. For this reason, it is critical that mature disciplines and processes be applied when working with software [Ahern et al., 2003]. Because software is nonphysical yet functionally behavioral, we suffer the challenge of attempting to measure certain nonmeasurable attributes. Nonetheless, that's all we can currently do to argue for the stability, functionality, and sustainability of the software over time [Voas, 2004].

Furthermore, software is developed or engineered, but it is not manufactured in the classical sense. As in hardware manufacturing, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) as compared to software. Both activities are certainly dependent on people (they are done by people!), but the relationship between the people assigned to the tasks and the work accomplished is entirely different. Both activities require the construction of a "product", but the approaches are different. Software costs are concentrated in engineering, and not for example in raw materials or heavy industrial equipments. Software does not "wear out". It deteriorates due to change: during its life, software will undergo change in which it is likely that errors will be introduced. Provided that there is no changes in requirement during the development, every software failure indicates an error in design or in the process through which design is translated into machine-executable code [Pressman, 2005].

As a human activity, programming involves and is influenced by human emotional elements, i.e. fun, boredom, etc, in the more apparent way in dealing with the bliss and the woe of the activity. An individual's

personality affects his/her ability to perform a particular job assignment [Cockburn, 2001].

Programming is fun because it gratifies creative longings built deep within us and delights sensibilities we have in common with all humans. Brooks (1995, p.7) describes the programmer in a beautiful and romantic way as follows: "the programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself."

Beside the joys, there are also the woes of programming. *Firstly*, the system produced is required to be, as much as possible, precise and error less. Therefore the programmer is supposed to perform perfectly. Human beings are not accustomed to being perfect, and few areas of human activities demand it. *Secondly*, one rarely controls the circumstances of one's work; most often a programmer depends upon other people's programs including the mal-designed, poorly implemented, incompletely delivered and poorly documented ones. *Thirdly*, finding bugs is not fun but it is inevitable. Debugging has a linear convergence: the last and difficult bugs usually take more time to find than the first. *Finally*, the pressure of time constraints: the product over which one has laboured so long appears to be obsolete upon (or before) completion [Brooks, 1995]. In line with this, people don't work through their problems in a nice and tidy fashion. Thus, legislating how a person is to solve problems often invites trouble [Cockburn, 2001].

Besides, there are also external pressures for the profession and the business in general such as the rapid development in the technology itself. Some reports suggest that the half-life —the time it takes for half an engineer's knowledge to become obsolete —is two to three years for software [Costlow, 2003].

Looking over more closely the way software practitioners and researchers view software, I propose three different dimensions in viewing software and its development process. Emphasizing one dimension over the others influences the way methodologies are tailored and eventually the way quality is perceived. First, it is the *product dimension* where software is mainly seen as a developed or engineered product. Second, it is the *process dimension* where software is perceived as shaped by its process and the only way to control and manage the outcome is by controlling and managing the process. Third, it is the *people dimension* where the roles of people (and its human characteristics) involved in the production are recognized and made central to the work setting and organization. In the real world,

these dimensions are not strictly detached. In most cases, methodologies need to be customized and tailored; and quality is often perceived based on the combination of these dimensions, with varying degrees of emphasis.

2.1.1 Product Dimension

Viewing software as a product, or a programming system product [Brooks, 1995], allows and encourages software practitioners to take measures similar to traditional manufacturing in ensuring its quality. The measures then are applied to the core ”engineering” tasks which develop the product, spanning from the analysis of software requirements at the front end of the development, to the software integration and testing at the concluding end. Software is usually built on or by using existing software products such as programming languages, development environments, etc., such that the base product influences the end result by imposing technical constraints. I argue that the product oriented approach tries to guide quality improvement by making product quality explicit. Raw material inputs, for example user requirements, should strictly conform to the characteristics compelled by the well-defined production process, i.e. methodologies, testing, etc., using several accurately measured tools, i.e. development tools and programming languages. In my point of view, applying this dimension strictly will simply dismiss the ”soft-ness” of software. Software will be no different than hardware. One will tend to impose artificial ”laws of physics” to the supposedly intangible and malleable entity, for example, by exaggerating formality in the entire process production and expecting such formality to enable repeatability of the process and result.

2.1.2 Process Dimension

The process dimension is recognized by those who argue that since software is intangible, malleable and deteriorates due to changes caused by mal-design, then the development of software pre-eminently requires mature disciplines and processes to ensure its quality. This approach does not emphasise formality for the sake of repeatability. It tends to be exhaustive and very careful about the processes to ensure that all possibilities of failure are avoided. It also relies on the quality of the process and believes that a quality product can only be achieved by going through the quality process. In other words, the process oriented approach tries to improve product quality indirectly, by controlling and improving the soft-

ware development process by assuming that there is a positive correlation between process improvement and product quality.

Many authors share the idea of seeing software as mainly a process; and it seems that this dimension is more popular, probably, due to its practical outcomes. The precise knowledge of software quality is usually not available until very late (usually during operations and maintenance) in the software life cycle. In contrast, software product and process metrics can be obtained relatively sooner than the software quality information [Khoshgoftaar et al., 2004]. Software process is a series of actions that should be taken in developing a software system, such as gathering the requirements, analyzing the requirements, specifying the system, designing the system, and so forth [Filman et al., 2004]. Most software processes, in turn, break down these high-level actions to successively smaller action granularities. Siakas (2002) also shares the argument that quality of the software product cannot be decided until late in the development cycle. During the 1990's, an increased consideration was given to the process used for software development and its potential to improve software quality. The popularity of CMM, ISO 9000 series of standards and the SPICE model are testament to the importance associated with a process-orientation for Software Engineering [Ward and Aurum, 2004].

2.1.3 People Dimension

Recent trends in software methodology as well as in the development of QMS recognize a new dimension in software by acknowledging and emphasizing the role of people, both as individuals and teams. As mentioned before, the relation between the software and its creator is so close that human elements become highly influential to the outcome.

Emphasis on the role of people in software development is argued by Cockburn (2001) who states:

> "The last time people constructed a vocabulary for software development was in the late 1960s, when they coined the phrase software engineering, as both a wish and a direction for the future. It is significant that at the same time the programming-should-be-engineering pronouncement was made, Gerald Weinberg was writing The Psychology of Computer Programming. In that book, software development does not look very much like an engineering discipline at all. It appears to be something very human-centric and communication-centric. Of the two, Weinberg's observations match what people have reported in

> the succeeding 30 years, and software engineering remains a wishful term. (p.xviii)"

Cockburn (2001) elaborates the elements in a light-hearted section labelled "elements of funkiness" as followed: humans are spontaneous, both for good and for bad; we are happily contradictory; we are stuffed full of personality which vary by hour, by day, by age, by culture, by temperature, by who else is in the room and so on.

This dimension is necessary to pay attention to especially when orchestrating a software development in a team in which people with different psychological traits and social backgrounds work together. Cockburn (2001) says that depending on almost anything, a person can be cooperative with one person at one moment and belligerent the next moment or with the next person.

Emphasising this dimension means focusing the "soft-ness" of software because human elements are those that make software different from traditional engineering. It emphasizes the uncertain and dynamic elements in the project management which affect the outcomes. It is the source of creativity and innovation as well as the potential for errors and conflicts. However, it seems that this dimension is the most difficult to maintain and manage.

Many people have recognized the importance of people's roles and interaction in software development, but little has been done to actually address the issue in a more concrete manner. Methodologies and approaches have been created to facilitate the human elements such as pragmatic programming, programming in pairs, and other practices suggested by agile software development. Besides, quality is not an objective entity. It is in fact an agreement between the stakeholders of the project, so again, the people dimension!

The Roles of Customer in Defining Quality

So far it has been admitted that there is no objective quality in software project. Many authors on quality, in general or specific for software, shift the focus to customers as having the primary role in determining the subjective and contextual quality of a product or a project. If beauty is in the eye of the beholder, then quality must be as well [Voas, 2004]. In his book, Lila, An Inquiry into Morals (Bantam, 1991), Robert Pirsig defines quality as value – to somebody. This broad but useful definition implies that each "somebody" will have different ideas about value. Whenever several people are involved with a software system, several different values

will apply to that system [Robertson, 1995]. In market-driven software business, customer wishes and product requirements are the two areas that software companies must find and maintain the links to survive and win the competition [Natt och Dag et al., 2005]. Attempts by several major commercial software vendors to improve software quality with better tools are potentially important, but tools will find only the more obvious coding problems, such as syntax errors and usage inconsistent with programming conventions [Vaughan-Nichols, 2003].

In the global outsourcing context, the importance of customer in defining quality is heightened. They are required to be more informed about what they want and to produce better product specifications. The difficulty is that requirements aren't about the software but about the problem world, which is always potentially complex, even for a system whose software is itself quite simple [Jackson, 2004]. "What offshore forces a company to do, is write a very detailed spec for a product," Jeffrey Tarter, a veteran software industry analyst and editor of the Softletter newsletter, says. "That form of discipline is incredibly valuable. It forces people to consider every screen, the flow of the product, and which features are necessary. This turns out to be why the Indian firms have been successful" [McLaughlin, 2003] p.115. On the vendor side, it is useful and necessary to facilitate the customer with methods that enable them to define the requirements.

Meeting the requirements might be different from being fit for a purpose, which can also be different from complying with the rules and regulations on how to develop and deploy the software. Yet we can think of all three perspectives as ways to determine how to judge and assess software quality [Voas, 2004]. Many people agree with the view that quality is to meet user requirements: a "high-quality" Web site is one that meets its owner's and users' requirements [Mich et al., 2003]. Most activities to improve software reliability focus on how developers can predict and prevent defects by analyzing a customer's view – the software's field failure rate [Wood, 2003]. Voas (2004) adds that software quality is nothing more than a recipe. Some like it hot, sweet, salty, acidic, or greasy. The end user is the restaurant patron. Season to taste.

The following section examines several widely known attempts to measure the quality of software.

2.2 Quality Measurement

This section discusses several approaches to measure quality in software. Being a good solution to the right problem is our definition of quality software so far. Let us scrutinize the development process further by examining the stakeholders involved in it. Software development is usually involving stakeholders with different priorities and concerns such as: *a project manager* who has the responsibility for the project on the supply side, *a business analyst* who has to liaise with the users and should understand their needs better than anyone else amongst the suppliers, *an implementation programmer* who writes the software, *a quality editor* who detects departures from a quality solution, *an end-user* who has to use the system in the end, *a line manager* who is the end-user's boss and may be the instigator of the project, and *a project sponsor* who pays the bill. Each stakeholder has his or her own objectives from the project, often conflicting with each other.

The overall quality may be judged in terms of how well each stakeholder is satisfied at the end of the day. In an attempt to classify different and possibly conflicting views of quality, Garvin (1984) has suggested five different views of quality: (1) The transcendent view which relates quality to innate excellence; (2) The product-based view which views that it costs money to build in quality which can be added to a product in two ways, namely, by putting greater functionality and by taking greater care in development which is leading to a higher quality solution; (3) The user-based view quality which is defined in terms of giving the users what they want or "fitness for purpose"; (4) The manufacturing view quality which is measured in terms of conformance to requirements; (5) The value-based view to provide what the customers require at a price they can afford [Garvin, 1984]. Further, Gillies (1997) adds another view that software quality is actually about people. Tools, process and quality management systems are all aids to enhancing quality, provided that the people are capable and motivated towards their effective use. The latter view emphasising the role of people is an important building block of this thesis and will relate to other aspects of software development.

There have been attempts to concretize the notion of quality in order to use it in practical situations. In order to compare quality in different situations, both qualitatively and quantitatively, people build *models of quality* which mainly deal with issues: what criteria of quality should be employed, how the criteria interrelate and how the associated metrics may be combined into a meaningful overall measure of quality. A *metric* is a measurable property which is an indicator of one or more of the quality crite-

ria. There are two types of software metrics: predictive and descriptive. A *predictive metric* is used to make predictions about the software later in the lifecycle, for example, structured ness is associated with maintainability of the software product in use. A *descriptive metric* describes the state of the software at the time of measurement, for example, reliability metric might be based upon the number of system "crashes" during a given period.

Many researchers have tried to study quality from various angles. Quality is measured by product failures in the field and user satisfaction is measured by customer feedback; and a conceptual model of software management is one that simultaneously considers development cost, quality and user satisfaction [Krishnan, 1993]. A study on the adaptation and use of the Quality Function Deployment (QFD) – which is an implementation vehicle of Total Quality Management (TQM) – for software development by a major software vendor shows that QFD can be useful as a preceding activity to Software Development Life Cycle processes as a front end requirement solicitation technique that quantifiably solicits and defines critical customer requirements [Haag et al., 1996]. Customer perceived quality can be measured by various services and the probability of observing a software failure can be increased by using the deployment schedule, hardware configuration and software platform [Mockus et al., 2005]. The quality of outsourced IT services can be traced to attributes of services such as tangibles, responsiveness, reliability, assurance and empathy; and it was proposed that transaction cost economics (TCE) characteristics – frequency, uncertainty, and asset specificity – of each service determine the drivers of client satisfaction for that service [Das et al., 1999].

The -ilities (or software attributes) are a collection of closely related behaviors that by themselves have little or no value to the end users but that can greatly increase a software application or system's value when added. Examples of these flavor-enhancing software -ilities include maintainability, reliability, usability, efficiency, adaptability, availability, security, portability, scalability, safety, fault tolerance, testability, usability, reusability, and sustainability [Voas, 2004]. The problem is that these attributes are *unlikely* to be associated in any way with how the software is developed since development standards seldom drill down low enough to provide guidance needed to obtain particular degrees of any of these attributes. That is why metrics and other measurement techniques are created: to try to monitor and quantify those attributes.

Now, an approach to achieve quality through the software engineering methodology is discussed in the following section.

2.3 Software Engineering Methodologies: Building Quality?

This section presents a literature review on software engineering methodologies and their common practices. The software engineering discipline has been dubbed as one of the approaches to the quality software development. As defined by the IEEE Standard 610.12, software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software – that is, the application of engineering techniques to software [Ahern et al., 2003].

2.3.1 Method or Methodology?

The development of systematic procedures to produce structured code, which became known as 'methodology', was the first widespread attempt to take account of quality issues during software development. The term 'methodology', which originally means the study of method, has passed into computing jargon and has come to mean a systematic framework for the development of software. In practice, it is often incorrectly used synonymously with the word 'method'. Avison & Fitzgerald (2003) argue that the 'philosophy' element helps to distinguish "methodology" from "method". The philosophy element refers to the underlying theories and assumptions that the authors of the methodology believe in and that have shaped the development of the methodology. They believe that the definition of a methodology should include specific reference to its philosophy as this has a critical bearing on the understanding of a particular methodology. Generally, a good methodology for software development has a number of characteristics such as: usability, integrity, adaptability to local needs, clarity, and automation.

Software engineering was developed to address the problem of handling complexity through a structured approach which is hoped to reduce costs through increased productivity during initial development by reducing errors. It was also intended to reduce ongoing maintenance costs through making software more reliable. The introduction of software engineering has clearly increased the quality of software in terms of technical parameters such as reliability and maintainability when measured in a narrow technical way, and in terms of 'conformance to specifications'. However, the underlying problem which software engineering was set up to address was the cost of software maintenance resulting from error correction, failure to meet users' initial needs, and failure to cope with evolv-

ing needs.

We can look closer to the development practices prescribed in methodologies available by using the software dimensions and show that some practices are more focused on one dimension than the others. This way of approaching those methodologies informs us both of their potential and pitfalls in delivering successful projects. The elements of a software methodology are presented in the subsection below.

Elements of Software Methodology

The elements of Software Methodology presented here referred to the work of Cockburn (2003).

- *"Roles"* refers to the position of the people employed in the team. An interesting aspect to assess is why and how a role is defined in a certain team.
- *"People"* here refers to the personality trait of each person expected for the roles in the team, i.e. a project manager should be good with people, a user interface designer should have natural visual talents, and some empathy for user behaviour, an object-oriented program designer should have good abstraction faculties, and a mentor should be good at explaining things.
- *"Skills"* refers to the personal prowess of a person needed for the roles, which is a product of his training and talent. One can assess how a team strives with skills which are not presently available.
- *Teams* are roles that work together under various circumstances. There may be only one team on a small project. On a large project, there are likely to be multiple, overlapping teams, some aimed at harnessing specific technologies and some aimed at steering the project or the system's architecture.
- *Tools* refer to software, hardware and other tools that make the development work possible.
- *Techniques* are the specific procedures people use to accomplish tasks. Some apply to a single person, i.e. writing a use case, managing by walking around, designing a class or test case, while others are aimed at groups of people, i.e. project retrospectives and group planning sessions.

- *Process* refers to how activities fit together over time, often with pre- and post-conditions for the activities.
- *Activities* refer to how the people spend their days, i.e. planning, programming, testing, and meeting.
- *Milestones* are events marking progress or completion. Some milestones are simply assertions that a task has been performed, and some involve the publication of documents or code. A milestone has two key characteristics: it occurs in an instant of time, and it is either fully met or not.
- *Work Products* are what one constructs such as design cards (disposable) or the usage manual or source code (relatively more permanent).
- *Standards* here refer to the conventions the team adopts for particular tools, work products, and decision policies.
- *Quality* measures refer to the degree of achievement of the activities or the work products.
- *Team values* are value systems which govern the rest of the methodology elements, for example, an aggressive team working on quick-to-market values will work differently than a group that values families and goes home at a regular time every night.

I do not suggest that every single methodology contains those elements. Some methodologies specify some of those elements more than the others. In the following section, several examples of software methodologies are presented.

2.3.2 Examples of Software Methodologies

In this section, examples of popular software development methodologies are presented – i.e. SDLC, evolutionary model (prototyping) and agile development – followed by a general classification of methodologies based on seven broad themes/approaches.

A. Software Development Life Cycle (SDLC)

SDLC was, and still is, the mainstream approach in software engineering. It has had an immense influence as a general approach to develop

software. Even though there are many variants, SDLC has the following basic structure: (1) feasibility study, (2) system investigation, (3) system analysis, (4) system design, (5) implementation, and (6) review and maintenance. These stages together are frequently referred to as 'conventional system analysis', 'traditional system analysis', 'information system development life cycle', or, more frequently in the USA, as the *waterfall* model. The term 'life cycle' indicates the staged and linear nature of the process. Further, by the time the review stages comes, the information system must be found to be inadequate and it may not be long before the process starts again with a feasibility study to develop another information system to replace it [Avison and Fitzgerald, 2003a].

Avison and Fitzgerald (2003a) noted the benefits of SDLC as follows. Methodologies incorporating this view of application development have been well tried and tested. The use of documentation standards in such methodologies helps to ensure that the specifications are complete, and that they are communicated to the system development staff, the users in the department, and the computer operations staff. It also ensures that these people are trained to use the system. The education of users on subjects such as the general use of computers is also recommended, and helps to dispel fears about the effects of computers. Following such a methodology also prevents, to some extent at least, missed cutover dates (the date when the system is due to become operational) and unexpectedly high costs and lower than expected benefits. At the end of each phase, the technologists and the users have an opportunity to review progress. By dividing the development of a system into phases, and further subdividing them into more manageable tasks, along with offering improved training and the techniques of communication, we have the opportunity for control of the application development process.

However, Avison and Fitzgerald (2003a) also noted the criticisms of SDLC, or, to be more precise, of the way in which it was used. These include: failure to meet the needs of management, instability, inflexibility, user dissatisfaction, problem with documentation, lack of control, incomplete systems, application backlog, maintenance overload, problems with the 'ideal' approach, and emphasis on 'hard' thinking. Some of the criticisms will be elaborated below.

Inflexibility was the result of an output-driven design. The outputs that the system is meant to produce are usually decided very early in the systems development process. Once the output is agreed, the inputs required, the rest of the systems which convert the inputs to the outputs can all be designed. However, changes to required outputs are frequent. And because the system has been designed from the outputs backwards, changes

in requirements usually necessitate a very large change to the system design. The changes therefore cause either a delay in the implementation schedule or are left undone, leading to an unsatisfactory and inappropriate system.

Documentation, even though it was mentioned as one of the benefits above, is not ideal. The orientation of the documentation is frequently towards the computer person and not the user, and this represents a potential source of problems. The main purpose of documentation is that of communication, and a technically-oriented document is not ideal. Documentation is often perceived as a time consuming and less creative activity.

Emphasizing 'hard' thinking, the SDLC approach may make a number of simplistic assumptions. It assumes that there are 'facts' that only need to be investigated and identified; it assumes that there can be a 'best solution' identified that will 'solve the problem'; it assumes that this 'ideal solution' can be easily engineered by following a step-by-step methodology; and, it assumes that the techniques offered will analyse and design all that needs to be done. We are not engineering a simple mechanical object. The world of information systems is concerned with organizations and people as much as technology. The situations encountered are often ambiguous, issue-laden, messy and problematical [Avison and Fitzgerald, 2003a].

It is interesting to note that after elaborating those criticisms, Avison and Fitzgerald (2003a) advice us to treat the criticisms as *potentials* since many organizations using the approach do not fall into all or even most of the potential traps.

B. Prototyping and Evolutionary Models

Prototyping is an approach based on an evolutionary view of software development, affecting the development process as a whole. Prototyping involves producing early working versions ("prototypes") of the future application system and experimenting with them. Prototyping provides a communication basis for discussions among all groups involved in the development process, especially between users and developers. In addition, prototyping enables us to adopt an approach to software construction based on experiment and experience [Lichter et al., 1993].

This approach has been particularly fruitful in participatory design, where users are brought in very early in the design phase. A representation can be a vehicle for communication, a tangible "placeholder" for the real thing, which represents a proposed artifacts role in a real-world situation. Designers and users can thus explore the user experience of a system together, before it actually exists – even if it is only represented by

an empty cardboard box [Holmquist, 2005].

Furthermore, Holmquist (2005) argues that a prototype represents the knowledge of *function*; it is a tangible artifact in which the necessary technology to achieve a particular functionality is implemented. However, the prototype says next to nothing about whether it will result in a successful product or systems. This is fine, the developer might argue, because these properties are separate from the function – they are part of the interface – and can be optimized now that the fundamental technical problem has been solved. By its very existence, the prototype constitutes an existence proof that a technology works.

As a representation, the prototype is a *generator*. A generator is at the center of a process that generates inspiration and ideas – it is not an end in itself. By making abstract thoughts concrete, and by providing a focus for exploration and discussion, a generator can give rise to new insights. What one should take away from a generator are ideas and inspiration, which are potentially valuable. However, the designer should be aware that this potential knowledge must be judged, validated and refined before being used or disseminated. Therefore, the value of representations as generators lies in how much the designer can ultimately take away from them, not the rigidity of the knowledge that went into creating them [Holmquist, 2005].

Three major activities of the software development process can be influenced by the construction of prototypes: starting the project, analyzing the business needs, designing and constructing the software system. To illustrate the way in which we see the relationship between prototype and these activities, Lichter et al (1993) distinguish between the following kinds of prototypes:

A *presentation prototype* supports the initiation of a software project. It is of major importance whenever an explicit contract is to be set up between a client and a software manufacturer. In most cases the presentation prototype is developed to show the users view of the envisaged system, i.e. it will present important aspects of the user interface. Furthermore, if the technical solution of a problem is unclear, the presentation prototype may present functional details to convince the customer of the possibility of solving his problem.

A *prototype proper* describes a provisional operational software system that is constructed parallel to the information system model. A prototype of this sort is generally designed to illustrate specific aspects of the user interface or part of the functionality and helps to clarify the problem in hand.

A prototype that is designed chiefly to help clarify construction-related

questions facing the development team is called a *breadboard*. A breadboard is derived from the information system model or the software specification. This kind of prototype is also encountered in traditional software projects, although the experimental approach associated with it is seldom given explicit recognition. Users are generally excluded from the evaluation of breadboards. To this extent, the use of breadboards is a restricted form of prototyping.

If a prototype is used not only for experimental testing of an idea or for "illustrative purposes," but is actually used in the application area itself as the core of the application system, it is known as a *pilot system*. In such cases, there ceases to be any strict distinction between the prototype and the application system. After reaching a certain degree of "sophistication" the prototype is implemented as a pilot system and enhanced in cycles.

Lichter et al (1993) distinguish between several different goals of prototyping: (1) *Exploratory Prototyping* is used where the problem in hand is unclear. Initial ideas are used as a basis for clarifying user and management requirements with respect to the future system; (2) *Experimental Prototyping* focuses on the technical implementation of a development goal. An essential aspect here is the communication between users and developers on technical problems and questions about software ergonomics; (3) *Evolutionary Prototyping* is a continuous process for adapting an application system to rapidly changing organizational constraints. This means that software development is no longer seen as a self-contained project, but as a process continuously accompanying the application.

C. Agile Software Development

Agile challenges the bureaucratic application of previous methodologies and the tendency of thinking that certain methodologies can be applied to any project in any given context. It is based on two core ideas: (1) different projects need different processes or methodologies and (2) focusing on skills, communication, and communities allows the project to be more effective and more agile than focusing on processes. Using vocabulary derived from the study of games, Agile Software Development supporters have tried to reformulate software development as "a (resource limited) cooperative game of invention and communication. The primary goal of the game is to deliver useful, working software. The secondary goal, the residue, is to set up for the next game. The next game may be to alter or replace the system or to create a neighbouring system" (p.31). While denying the extra burden from conventional methodologies, they consider methodologies to still serve several uses, namely: (a) introducing

new people to the process, (b) substituting people, (c) delineating responsibilities, (d) impressing sponsors, and, (e) demonstrating visible progress [Cockburn, 2001].

In Agile Development, its supporters re-examine the meanings of methodology and *prescribe to tailor a methodology for each project*. Quoting Maier and Rechtin (2000), there are four categories of a methodology: (1) *normative* – based on solution or sequences of steps known to work for the discipline, (2) *rational* – based on method and technique, (3) *participative* – stakeholder based and capture aspects of customer involvement, and, (4) *heuristic* – based on lessons learned. As a body of knowledge grows, sections of the methodology move from heuristic to normative and become codified as standard solutions for standard problems. Cockburn (2001) argues that most of software development is still in the stage where heuristic methodologies are appropriate. He identifies four challenges in designing methodologies, namely: variations in people, variations across projects, long debug cycles, and changing technologies.

Cockburn (2001) also states seven principles useful in designing and evaluating methodologies: (1) interactive, face-to-face communication is the cheapest and fastest channel for exchanging information, (2) excess methodology weight is costly, (3) larger teams need heavier methodologies, (4) greater ceremony is appropriate for projects with greater criticality, (5) increasing feedback and communication reduces the need for the intermediate deliverables, (6) discipline, skills, and understanding counter processes, formality, and documentation, and (7) efficiency is expendable in non-bottleneck activities.

In my opinion, Agile Software Development is not exclusively separated from traditional methodologies like SDLC. They, in fact, with their multiple methodologies principles, benefit from the availability of numerous "non-agile" methodologies. This movement, I believe, is simply to try to change the attitude towards software development by emphasizing the role of people. Software is not merely about a product or process, but it is about people doing the appropriate process which yields the desired product.

Quality Practices in Agile Software Development

Huo et al (2004) specifically address the quality aspects in Agile methodology. They identify Agile practices that have quality assurance "potentials", including:

- *System metaphor* is used instead of a formal architecture. It presents

a simple shared story of how the system works; this story typically involves a handful of classes and patterns that shape the core flow of the system being built. There are two main purposes for the metaphor. The first is communication. It bridges the gap between developers and users to ensure an easier time in discussion and in providing examples. The second purpose is that the metaphor contributes to the team's development of a software architecture. This practice helps the team in architecture evaluation by increasing communication between team members and users.

- Having an *On-site customer* is a general practice in most agile methods. Customers help developers refine and correct requirements. The customer should support the development team throughout the entire development process.

- *Pair programming* means two programmers continuously working on the same code. Pair programming can improve design quality and reduce defects.

- *Refactoring* is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. Its heart is a series of small behavior preserving transformations. Each transformation (called a 'refactoring') does little, but a sequence of transformations can produce a significant restructuring.

- *Continuous integration*, a popular practice among agile methods means the team does not integrate the code once or twice. Instead the team needs to keep the system fully integrated at all times. Integration may occur several times a day. "The key point is that continuous integration catches enough bugs to be worth the cost". Continuous integration reduces time that people spend on searching for bugs and allows detection of compatibility problems early.

- *Acceptance testing* is carried out after all unit test cases have been passed. This activity is a dynamic QA technique. A Waterfall approach includes acceptance testing but the difference between agile acceptance testing and traditional acceptance testing is that acceptance testing occurs much earlier and more frequently in an agile development; it is not only done once.

Early customer feedback is one of the most valuable characteristics of agile methods. The short release and moving quickly to a development phase

enables a team to get customer feedback as early as possible, which provides very valuable information for the development team [Huo et al., 2004].

QA techniques can be categorized into two types, static and dynamic. The selection, objectives, and organization of a particular technique depend on the requirements and nature of the project and selection is based on very different criteria depending on the methodology used. Static techniques do not involve the execution of code. Static techniques involve examination of documentation by individuals or groups. This examination is assisted by software tools, e.g., inspection of the requirements specification and technical reviews of the code [Huo et al., 2004].

Comparison of waterfall and agile methods has been done in literature. For example, Huo et al (2004) says that the waterfall model uses both static and dynamic techniques, while agile methods mostly use dynamic techniques.

Presenting Agile methods in this thesis serves several purposes including illustrating the various attempts and recent innovations in methodology world; to make comparisons with the established traditional waterfall model; and to enable us to see the possibility of both approaches in the global outsourcing context.

D. Other Methodologies

Since covering a broad range of the available methodologies is beyond the scope of this thesis, here, classification of methodologies is presented as a brief overview. Avison & Fitzgerald (2003b) classifies methodologies into seven broad themes, and approaches:

Structured. The concepts of structured programming were applied to analysis and design, and techniques (such as data flow diagramming) enabled the topdown analysis and representation of complex processes.

Data-oriented. The focus was understanding data as the key element in a systems development, and the most important technique was entity-relationship modeling.

Prototyping. The emphasis was building an approximation or representation of the system, allowing users to visualize and respond to it prior to its physical implementation.

Object Orientation (OO). The identification of objects and attributes (in whole or part) and classes helped provide the theoretical benefits of inheritance and reuse.

Participative. The crucial feature was involvement of users and other stakeholders.

Strategic. The emphasis was the planning of information systems and development of an information systems strategy to support and enable the overall objectives of the business; business process reengineering is sometimes viewed as a development approach focusing on strategy.

Systems. The complexities of human activity systems were addressed by adopting a holistic view far beyond a systems single-application boundaries.

2.3.3 Common Practices in Software Development

In this section, common practices in software development are presented. These practices are selected because they are part of various software methodologies; and they are particularly influential in quality achievement.

A. Software Testing

One widely used process that supports the construction of quality software is testing, which executes the program with input data or test cases, and then compares the output data to expected results [Malloy and Voas, 2004]. Software testing is to reveal bugs. Its role is limited purely to identifying bugs, not diagnosing or correcting them (debugging) [Binder, 2000]. The test cases specify the state of the code being tested and its environment, the test inputs or conditions, and the expected results. A *test suite* is a collection of test cases, typically related by a testing goal or implementation dependency. A *test run* is an execution of a test suite with its results. *Coverage* is the percentage of elements required by a test strategy that have been traversed by a given test suite. Regression testing occurs when tests are rerun to ensure that the system doesn't not regress after a change [Bruegge and Dutoit, 2004]. In other words, the system passes all the tests it did before the change [Crowther and Clarke, 2005].

Software testing practices consist of verification & validation (V & V) testing and defect testing, and reflecting on the two dimensions. V & V refers to the checking and analysis processes that ensures that software conforms to its specification and meets the customers' needs. It is a whole life-cycle process. The goal of defect testing is to expose latent defects in a software system before the system is delivered. This contrasts with validation testing which is intended to demonstrate that a system meets its specification. Validation testing requires the system to perform correctly using given acceptance test cases. A successful defect test is a test which causes the system to perform incorrectly and hence exposes a defect. This emphasizes an important fact about testing. It demonstrates the presence,

not the absence, of program faults [Sommerville, 2004]. V & V is based on the viewpoint that quality software is the result of a quality process which is ensured by the testing process in the whole life-cycle, while defect testing treats the software as a product to find its faults and errors.

Cleanroom software development is an example of V & V testing. It refers to a software development philosophy that is based on avoiding software defects by using a rigorous inspection process. The objective of this approach to software development is to produce zero-defect software. The name 'Cleanroom' was derived by analogy with semiconductor fabrication units. In these units (cleanrooms), defects are avoided by manufacturing in an ultra-clean atmosphere [Sommerville, 2004].

Functional or black-box testing is an example of defect testing. It is an approach to testing where the tests are derived from the program or component specifications. The system is a 'black box' whose behaviour can only be determined by studying its inputs and the related outputs. Another name for this is functional testing because the tester is only concerned with the functionality and not the implementation of the software [Sommerville, 2004]. Another approach to testing is called white-box testing (sometimes called glass-box testing). It is a test case design philosophy that uses the control structure described as a part of component-level design to derive test-cases. Using white-box testing methods, the software engineer can derive test cases that (1) guarantee all independent paths within a module have been exercised at least once, (2) exercise all logical decisions on their true and false sides, (3) execute all loops at their boundaries and within their operational bounds, and, (4) exercise internal data structures to ensure their validity [Pressman, 2005].

B. Bug-fixing

Following testing is the debugging (or bug-fixing) practice. A bug is, by definition, an unknown quantity [Dibling, 2005]. It is identified as a consequence of successful testing, that is, when a test case uncovers an error, debugging is an action that results in the removal of the error. Although debugging can and should be an orderly process, it is still very much an art. A software engineer, evaluating the results of a test, is often confronted with a "symptomatic" indication of a software problem. That is, the external manifestation of the error and the internal cause of the error may have no obvious relationship to one another. The poorly understood mental process that connects a symptom to a cause is debugging. Commenting on the human aspects of debugging, as argued by Pressman (2005):

> "Debugging is one of the more frustrating parts of programming. It has elements of problem solving or brain teasers, coupled with the annoying recognition that you have made a mistake. Heightened anxiety and the unwillingness to accept the possibility of errors increases the task difficulty. Fortunately, there a great sigh of relief and a lessening of tension when the bug is ultimately corrected. (p.413)"

The last remark signifies the people dimension of software, especially the psychological or emotional challenges one must undergo to accomplish the task. There are several strategies and tactics in undertaking debugging, namely: brute force, backtracking, cause elimination and binary partitioning. Each of these debugging approaches can be supplemented with debugging tools that provide semi-automated support [Pressman, 2005].

C. Software Documentation

Another practice required from software developers is documentation. Documentation standards in a software project that are particularly important as documents are *the only tangible way of representing the software and the software process*. Standardized documents have a consistent appearance, structure and quality and should therefore be easier to read and understand. There are three types of documentation standards: documentation process standards which define the process which should be followed for document production; document standards which govern the structure and presentation of documents; and, document interchange standards which ensure that all electronic copies of documents are compatible. Documentation includes giving comments embedded in the source code so that other developers are able to continue and maintain it, should the first author of the code is no longer available. This approach is believed to improve quality: "Another alternative to improve software quality is by using assertion while programming – a formal constraint on the behavior of a software application, usually written as an annotation describing what the application is supposed to do" [Malloy and Voas, 2004]. In general, documentation is not the most favourable thing to do, especially for software developers because it simply adds pressure to the already stressful condition of their work, and it divides their time from things they consider important – programming.

2.4 Quality Management System (QMS)

There are four principal aspects to a QMS for software development: (1) development procedures which includes the use of design and development methodologies and tools, testing and associated staff training; (2) quality control which includes many activities for the monitoring of quality during development; (3) quality improvement which includes all activities aimed at establishing a human quality culture amongst the staff; and (4) quality assurance which becomes the monitoring of the quality system itself to ensure that it is being carried out correctly.

Elements of Quality Management System

Drawing from Gillies (1997) I provide a description of the various elements of the QMS:

- *Organizational Structure* refers to the mechanisms in the organization or in the team which is assigned responsibility for quality.
- *Responsibilities* refer to the area of certain quality achievement intended and the actor to whom that achievement is assigned.
- *Procedures* refer to sets of activities to be undertaken to achieve quality objectives.
- *Processes* refer to individual activities that can be combined to form quality procedures.
- *Resources* refer to supplies of human, hardware, software or other supports while conducting quality procedures.

Several standards are available for quality management and process improvement such as ISO9000 and CMM. In the following sections those two widely-used standards are discussed.

2.4.1 ISO 9000 series

Standards are generally defined in terms of a model of best practice, against which all others may be compared. The ISO9000 series international standards were defined for quality management systems in 1979 and modified considerably in 1994. Among the standards, ISO9001 is generally the standard applied within software development because it is intended for

application where there is a significant design element. The standard is intended to be realistic and implementable and, therefore, sets no prescriptive quality performance targets, referring instead to standards agreed as part of the contract with the customer and acceptable to them. The standard focuses upon ensuring that procedures are carried out and results are documented in a systematic manner.

Generally, the changing process to comply with a certain standard requires considerable efforts. Implementing Software Quality Assurance in different types of organizations is a difficult and expensive task, requiring another way of thinking and more activities to be performed by the employees than they are used to. Therefore, guidelines for careful adaptation are required [Hosny, 2004]. However, the main concern of the ISO standards is that they do not provide a complete solution. Being prescriptive, it does not specify any particular tools or methods at any stage; it does not require any specific levels of performance, merely that an appropriate level of performance is achieved consistently.

2.4.2 Capability Maturity Model

Since its initiation in 1986 – when Watts Humphrey, the SEI, and the Mitre Corporation responded to a request by the U.S. federal government to create a way of evaluating the software capability of its contractors – the CMM has been intended to help software organizations improve along an evolutionary path, growing from an ad hoc, chaotic environment toward mature, disciplined software processes. SEI's CMM has evolved into CMM Integrated (CMMI) in which CMM for software process is still included. CMMI consists of three source model disciplines, namely: Software (SW-CMM), Software Engineering, and Integrated Product and Process Development. A fundamental choice faces the user of a Capability Maturity Model Integration (CMMI) model today is whether you use the continuous or staged representation. The question relates to the architectural structure of the model. One source model for CMMI, the CMM for Software, is a "staged" model. Another source model, the Systems Engineering Capability Model, is a "continuous" model. The third source, the Integrated Product Development (IPD) CMM, is a "hybrid" that combines features of both the staged and continuous

The maturity dimension of a CMMI model is described in the staged representation. Five levels of maturity exist, each of which indicates the process maturity of the organization. Figure 2.1 below shows the maturity levels (MLs) and their characteristics.

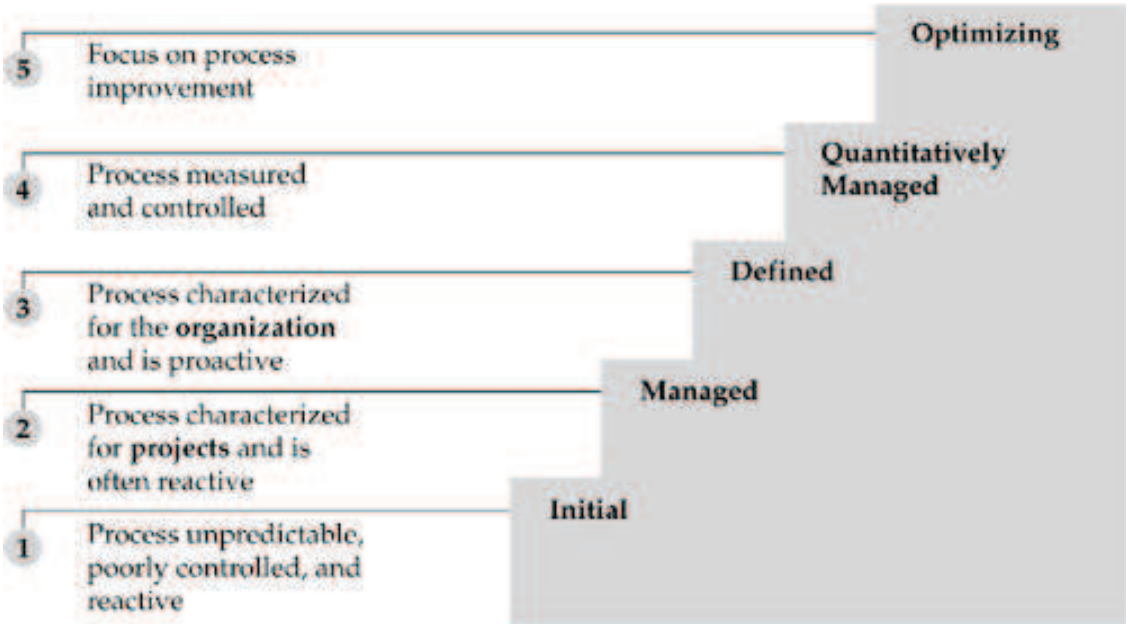

Figure 2.1: CMM Staged Maturity Levels

CMMI models consists of materials which help you both evaluate the content of your process and improve process performance. CMMI provides guidance for managerial processes. [Ahern et al., 2003]. The improvement information in CMMI models includes the creation of a viable, improvable process infrastructure. To build this infrastructure, CMMI includes ways to get your organization to focus more on defining and following its processes, such as training and standardization, and the use of measurement data to improve the process performance, innovate when processes need to evolve, and ensure the ability to meet changing needs.

The fundamental organizational feature of all the CMMI models is the ”process area.” Not everything related to processes and process improvement are included in a process-improvement model. Like its predecessor models, CMMI selects only the most important topics for process improvement and then groups those topics into ”areas.” The Software CMM focused primarily on process management. Among the ”key process areas” in the model, only one specifically targets the core engineering tasks, which range from the analysis of software requirements at the front end of development, to software integration and testing at the concluding end. This key process area was called Software Product Engineering in the Software CMM, and it dealt with tasks specific to software development. All of the other SW-CMM key process areas were written such that they could

easily be applied to development work other than software. Along with the success of the Software CMM in improving software development, this flexibility may explain the interest in applying the CMM concepts to disciplines beyond software; much of what was believed to be good with the Software CMM had a utility that was not restricted to just the software area.

The software industry has adopted CMMI since early inception for many reasons. Focusing on what the customer wants, rather than spending time fixing defects, inventing process, or perfecting tasks that are unnecessary in the first place can be achieved in CMM level-5 organization. Creativity is channeled into more effective and productive areas in high maturity organization [Eickelman, 2003].

Now, that we have examined various approaches to software quality, the next section presents a proposed quality framework which is then used in the case study analysis.

2.5 Proposed Quality Framework

In this section, I propose a quality framework which takes into account some of the approaches and criticisms of the current software methodologies and QMS, as well as the complexity and uniqueness of GSW. The next subsection discusses the need for a framework, followed by a discussion on its philosophical underpinning and an introduction to the framework itself.

2.5.1 The Need for a Framework

Software development is a complex process, and conducting it in GSW context makes it even more so. Observing quality in an ongoing GSW is practically impossible. Furthermore, little has been done to examine quality in GSW context in particular, which is so far believed to be different from the traditional co-located development. Therefore, we need a framework to help us monitor the various elements involved in software development projects which yield quality products. In doing so, we examine the well-established approaches around software and quality (i.e. software methodologies and QMS) which inspire the development practices; and then try to see the intricate connections between the elements of those approaches in action. The interrelations between those invariant elements are possibly unique and dynamic in each project, and should be treated as a different category of elements in the framework. Finally, the

two categories of elements (invariant and dynamic) are compiled in a single quality framework. The framework includes different quality stages in which the dynamic elements are intertwined throughout the projects. Complete description of the framework is presented later in this section.

2.5.2 Philosophical Underpinning

There are various elements of quality which are *relative* – they mean different things in different situations – and unique for various events in a project as well as in the organization. These uncertain events often co-exist with the more *invariant* activities for which metrics and methodologies are more useful to apply. The invariant elements in software development are mostly derived from its product and process dimension – such as methods, tools and processes – which have been the basis of many current development methodologies. They are common and familiar because they have been widely practiced. Besides, that approach has also been the foundation for education and training new software practitioners. We can thus safely assume that everyone involved in a software project, at least has an idea what software is and what are the common practices in software development like requirement engineering, design, testing, documentation, project management, etc. This knowledge enables them to coordinate and communicate, otherwise it will be impossible to conduct a software development.

On the contrary, the uncertain elements are usually derived from the people dimension of software involving activities such as communication and coordination between people involved in the development. Therefore, these elements should be handled with greater care in GSW due to the inherent challenges originated from the time, space and cultural separation. They have to be specifically tailored for each individual project.

The use of methodology has been problematic. For many organizations, adoption of a methodology has not always worked out or been the success its advocates touted. Real-world performance has led some developers to reject methodologies in general terms and attack the concepts (such as step-by-step development and meticulous documentation) on which they are based. Many reasons for this developer backlash have been suggested. The most common is probably disappointing productivity; another is that the methodologies are overly complex, usually designed for the largest and most complex development projects. They may lead to developing requirements to the ultimate degree, often over and above what is legitimately required, sometimes encouraging users to cre-

ate unrealistic wish lists. They also require highly technical skills that can be difficult and expensive for developers and end users to learn or acquire. Moreover, the tools advocated by methodology proponents can be costly, difficult to use, yet still not deliver enough benefit [Avison and Fitzgerald, 2003b] p.81.

Furthermore, Avison & Fitzgerald (2003b) argue that methodologies are often not contingent on the type or size of a project, nor upon the technology environment and organizational context. A methodology is often said to be *one-dimensional*, that is, it adopts only one approach to the development of projects that may well not address a particular organizations underlying issues or problems. Few recognize or address the critically important social, political, and organizational dimensions of development.

In the proposed framework I try to incorporate the three-dimensional perspective of software in the greater picture of GSW. I argue that people dimension is the primary source for the relative quality elements which are to be captured in the *quality stages*. Whereas, the other two dimensions – process and product – provide the invariant elements as the minimal requirement to work as a team, referred to in the framework as *the source for quality practices* (see fig. 2.2). Related to the people dimension, in GSO clients are usually separated by time, geographical and cultural diffences. This fact plays an important role in the variable quality area. Project teams along with the clients are forced in a novel way to identify what they want, analyse its feasibility, determine how to achieve it, and therefore, identify the criteria for success.

This framework is proposed to serve the following objectives. *Firstly*, it should map out the practices derived from, or advised by, methodologies and QMS in those quality stages. *Secondly*, it should indicate in which phases quality practice is highly, moderately or least critical. It may capture the differences of the phases in the GSW context in terms of its criticality to the quality achievement. I suspect that some phases are probably more critical than their corresponding ones in co-located development. *Thirdly*, it should show varying emphasis on both the human and cultural factors as well as the technical ones in the various stages of the development process. Some of the processes in software development are more human-oriented than others. For example, requirement gathering is more human-intensive than coding because with detailed and complete requirements, coding can become relatively a simple task of translating these requirements into machine language. This task can be made possible by the use of standards, programming conventions, and reuse of components. The creative, or to be more precise, the uncertain part is more prevalent in the analysis and design phase. *Finally*, it should show the proportion of

emphasis on the different software dimensions (product, process & people) at individual phases.

2.5.3 The Framework

The framework is intended to describe how the software team strives to achieve the quality goal. This framework consists of two primary components, namely: the *source for quality practices* and the *quality stages*. This framework also embraces the fact that it is hard to define software quality, but it is possible to recognize the presence and, even more so, the absence of quality. Therefore it collects all possible approaches which are aimed to solicit the presence – and consequently avoid the absence – of quality at any given time in the project. Quality practices refer to activities derived from the available development methodologies and QMS, which are regarded here as the more invariant elements of the framework. Quality stages refer to the phases in the development process in which those invariant elements are combined and managed to achieve quality goals. This part of the framework, I believe, represents the more relative and unique elements in a software development project. The framework is illustrated with the figure 2.2 below, followed by a more detail description of the elements of the framework.

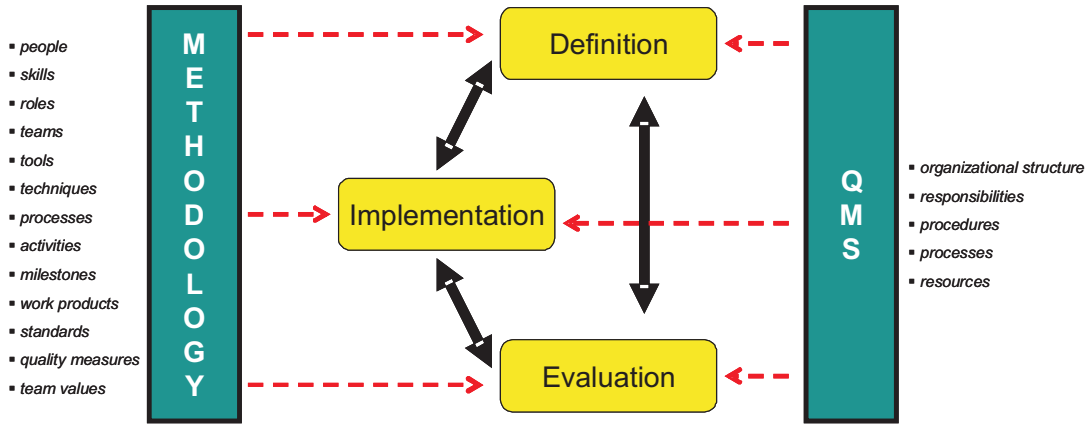

Figure 2.2: Quality Framework

Quality Practices

Quality practices are derived from the two main streams of software quality approach, i.e. software engineering methodology and QMS, as de-

scribed in the previous sections. A methodology is a description of conventions of interaction between roles [Cockburn, 2003], and of the way to achieve each action in a software process [Filman et al., 2004]. In this thesis I refer to a methodology as suggested by Cockburn (2003) which covers the following elements: people, skills, roles, teams, tools, techniques, processes, activities, milestones, work products, standards, quality measures, and team values (See 2.3.2). QMS consists of organizational structure, responsibilities, procedures, processes and resources for implementing quality management (See 2.4.1). QMS is complementary to methodology since it ensures that the procedures laid down by methodologies are actually carried out to the required standards. The scope of the QMS is the organization, while methodology is the software project.

Quality Stages

The framework identifies three broad quality stages: definition, implementation, and evaluation. Each stage is mapped into the software developmental phases based on the prevalent common practices, i.e. definition stage is mapped to analysis and design, which is mainly for simplification. By doing so, I fully realise that it emphasises the product dimension with its ”hard” thinking approach. This is inevitable since it is the dominant approach at the moment. However, we can benefit from its intuitiveness and simplistic assumptions. To lessen the strict ”step-by-step” approach, as illustrated in fig. 2.2, the stages may be applied sequentially (static) or iteratively (dynamic) depending on the approach of the project, whether it is formal or agile. For either case we can try to adjust the quality stages with the corresponding development processes that we use. The stages are elaborated below.

- **Quality Definition.** This stage covers the various events from the initial contact – between the customer and the organization or the project team – until the detailed design document is agreed by both parties and sent for construction. At the organization level, the interaction may start earlier when the customer selects vendors by looking at their presentations (advertisement, website, etc), at which point the customer defines their own quality expectation and matching it with the vendor’s cost, profile and reputation. Quality definition starts here during the process of matching the expectations and the offer.

  Conversely, when the customer sends their initial requirements the vendor will consider it and match it with the available resources, at

which point the company considers the quality level that it can offer accordingly. Appropriately matching the needs and the available skills on hand determines the success of the project and therefore, also the quality of the products.

In this phase, there is intensive interaction between customers and the project team, which helps shape the mutual understanding of the problem (requirement analysis) as well as the proposed solution (design). This mutual understanding, in turn, becomes the basis for the mutual agreement on quality definition. The phase ends when the definition is agreed and ready to be implemented.

- **Quality Implementation.** This phase begins when the quality definition is available and lasts until the product delivery for acceptance testing. In this phase, all the plans previously made, for example in the form of analysis and design documents, are executed to produce the desired product.

  Communication with client may happen with less intensity, for example, only when something unforeseen happens or when there is a change of requirement. It all depends on how good, concrete and final the quality definition is. In the outsourcing context, the activities in the phase mainly take place on the provider side. Therefore, the competence of the project team at the client side to execute the developing task with designated tools and methods plays an essential role in determining the quality of the outcomes. After all, it is always prudent to involve the customer in the process by making the whole process visible through status reports and occasional meetings and/or communication.

- **Quality Evaluation.** This phase begins when the product is finished and sent to the customer for acceptance testing and ends along with the project sign off. Signing-off indicates the end of the temporary contractual relationship between the customer and the project team. This phase is important because it is actually the only time one can know for sure the quality of software, i.e. by judging the available finished product. Moreover, the customer's responses to the software developed are crucial in judging whether the product reaches its quality goal or not. The former represents a more objective assesment to quality, while the latter is the relative element drawn from the people dimension (customer's satisfaction).

Chapter 3

Research Methodology and Empirical Setting

In this chapter, I present the research methodology and the empirical setting on which this thesis is based. It starts with the research problem specification in section 3.1, including the research objectives, the personal motivation and perspectives, and the research questions. Section 3.2 introduces the interpretive research approach used in this thesis. Section 3.3 describes the empirical research setting and activities. Section 3.4 presents a brief account on the research experience. Finally, section 3.5 discusses the challenges and limitation of this research.

3.1 Research Problem

3.1.1 Objectives

This thesis attempts to contribute to the *understanding* and *finding of best approaches* in developing software quality in the GSW context. As previously mentioned, I am interested in identifying and describing different determinants of software quality in distributed development by analysing software outsourcing projects closely. Two existing software quality approaches, i.e. software methodologies and quality management standards, will be the starting point of the pursuit. According to the project presentations, the organization believes that the two selected projects here are successful – no complaints from the customers; therefore, we can learn what criteria are applied and how the project team managed to deliver given the constraints. I also want to explore the possibilities and/or urgency of either creating a new, context-specific methodological approach or devel-

oping a new standard that accommodates the constraints faced by GSWs. It would be useful to know what can and should be prescribed to software practitioners to overcome such challenges. A more personal perspective on the matter is presented in the following section.

3.1.2 Personal Motivation & Perspective

By working on this thesis I learnt about how to conduct research as much as the content of the research itself. I started timidly with a vague idea about outsourcing and software engineering which mostly I had got from theories and lectures in the university. My primary motivation was to somehow get the theories I learned to be useful in practice. Early on, as a student trained mostly in the technical discipline of computer science, I realized that there should be a bridge between the academic approach to outsourcing phenomena and the practitioners approach; practitioners here referred specifically to the project managers and developers. Most lessons I learned, described the phenomena of GSO, recognized and analysed some problems in it without actually prescribing something to solve concrete technical problems – at least, it was from my humble point of view. I naïvely desired that "something concrete". That was the reason I took the angle of software methodologies in the context of outsourcing, which I felt concrete enough to serve my motivation. My problem then was to find a good entry point for my cause.

I narrowed down the scope of my general interest in software engineering to software development in outsourcing context for several reasons. *Firstly*, as a student coming from a developing country (Indonesia), I see software outsourcing as a good opportunity for such countries to take part in the bigger scheme of the IT world. Sloper (2004) argues that global demand has enabled several developing nations to transform key sectors of their economies – something many others aspire to emulate. Software is leading the push to global job markets, creating a boon for emerging countries. A number of large corporations have set up development sites around the globe, making it possible to develop code around the clock. These companies arguably can hire talent wherever it exists [Costlow, 2003]. As a high technology, IT is relatively inexpensive to develop in terms of physical infrastructure, compared to, for example, automobile or aeronautics industries. It is also people intensive. Countries that can provide IT outsourcing imply having good potential in human resources. Developing countries are in an unfortunate position if they have to compete from scratch in terms of innovation in IT. By doing outsourc-

ing, in a way, they learn by doing. *Secondly*, the more I study about distributed software development, the more I imagine that it would be the mode of working in the future, at least in the IT-related industry. Not only in the context of outsourcing, but it can also happen in different working areas, for example, in multinational companies. The outsourcing phenomenon seems to keep growing. Recent news suggest that Europe is finally jumping on the outsourcing bandwagon by, as noted by Consultancy TPI of Woodland Tex., having awarded $ 41 billion in outsourcing contracts in 2004, nearly twice the level of two years earlier. One of the hottest nearby destinations for the work is Spain, where thousands of young Europeans flock every year in search of sun and sangria – and jobs [BusinessWeek, 2005].

Choosing quality as the research topic serves a twofold purpose. *Firstly*, quality has been an unfulfilled goal for software development in general. New methodologies are constantly created and QMS is being promoted to ensure the quality achievement without a strong conviction of the results. No methodologies can guarantee success to the project implementing it. Software process improvement communities recognize that there is still little knowledge of how prescribed methodologies and QMS might actually result in the software production. *Secondly*, quality is, and should be, the main concern of the outsourcing providers even more. Technically speaking, it is more challenging to develop software in an outsourcing context. The reasons have been elaborated in other parts of this thesis (See 1.2). Moreover, quality is important and necessary for survival in the more competitive and global business world. Outsourcing relationships are volatile. A competitive price cannot forever hold the relationship with customers. Many people share this point of view that outsourcing needs to be seen as a strategic management option, not just a way to cut costs [La Ferla, 2004]. Laplante et al (2004) argue against the myth that outsourcing is cheap. Even in India – where the perceived difference in relative economy would suggest a lower labor cost – based on their collective experience, the cost of a skilled developer is approximately estimated at $25 per hour [Laplante et al., 2004].

The preliminary problem formulation, created before undertaking the fieldwork, is illustrated in the following figure 3.1. I consider it useful to show the development of the idea at that time.

Previously, I was suspecting that there was only one source of developing quality software namely the Software Engineering discipline. I intended to study the way software project teams derived their practices, which I thought at that time as "simply" selecting the *tools*, *methods*, and *processes* to develop a software solution to a given problem, complemented

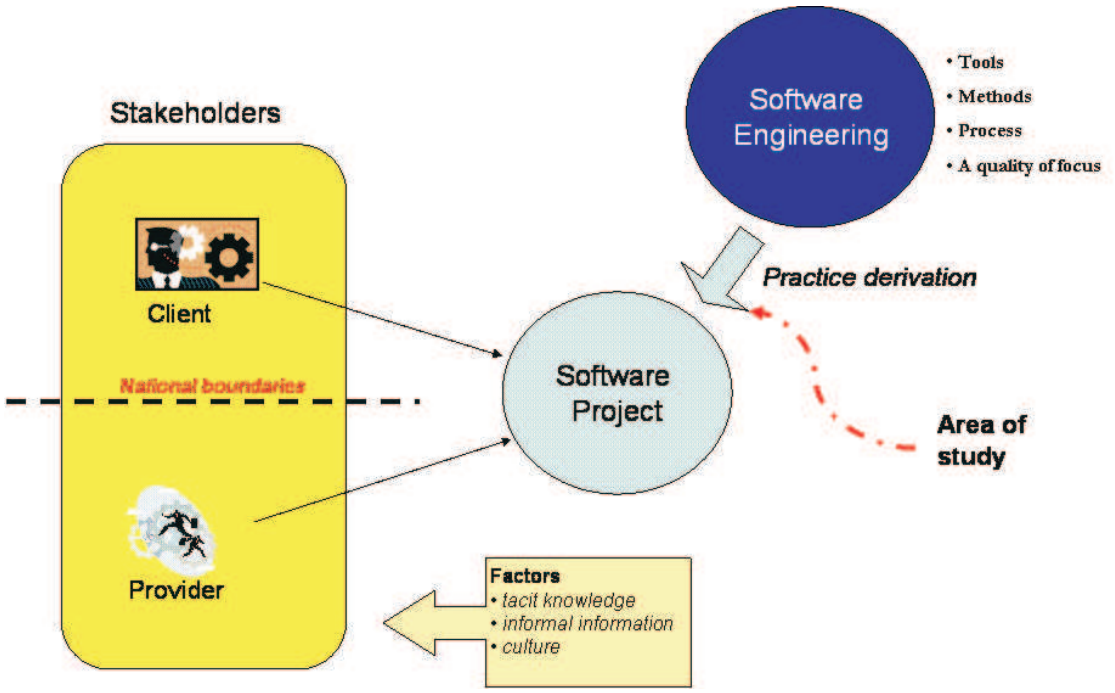

Figure 3.1: Early problem formulation

with a vague concept of *quality focus*. This model was also trying to capture the challenges in an outsourcing context, which by then was formulated as the *influential factors* of the provider, i.e. tacit knowledge, informal information and culture. This model turned out to be hard to relate to the area of study and the quality concept.

As the study went on, the early model seemed inadequate to capture the reality of the problem. Now, many people realize that Software Engineering has not fulfilled its promise of delivering "good" software and also that software is not merely about a product like other traditional manufacturing products. Software Engineering is, in fact, not the only path to better process and product. It is also difficult to see the hard evidence of the influential factors (tacit knowledge, informal information and culture) mentioned in the model in the technical level of analysis. Those factors may be more obvious in the higher level of analysis, such as the study of managerial issues or alliances between companies. All these lead to a new problem formulation presented below (fig. 3.2).

In this model, a software project is not seen merely as a single, flat, monolithic entity, but as a three dimensional one: product, process and people. These dimensions can help to make sense of the latest trends of

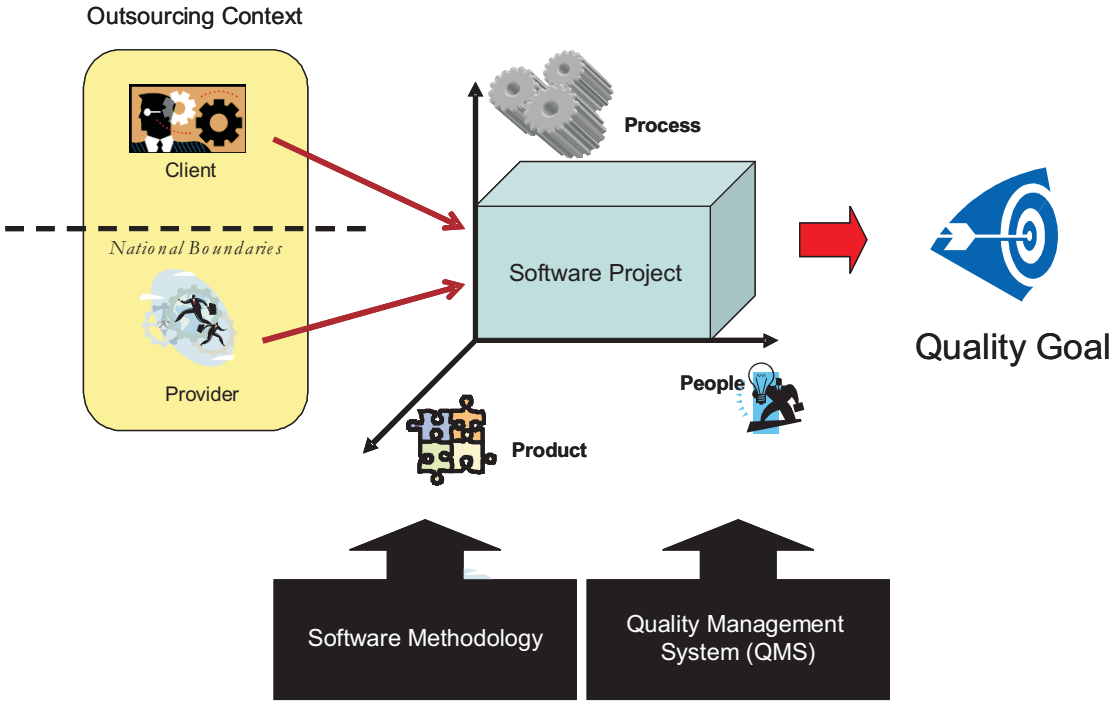

Figure 3.2: Current problem formulation

agile software development which emphasizes the role of people and the fact that no single methodology can be applied to all projects. Each project has its own combination of the people assigned, the selected process and the nature of the product desired. Some elements of methodologies can be tailored for each project because they are somewhat unique – at least one dimension of it is unique: the people. This opens the door to the investigation of different determinants of quality in the outsourcing context since the work arrangement involves people from different cultural backgrounds. Furthermore, this model recognizes two main strands of quality approach in software development, namely: Software Methodology (or Engineering, some might say) and Quality Management System. These two approaches are complementary by nature, since the former is prescriptive while the latter is enforcing the implementation of the former. These arguments led to the formulation of my research questions which are articulated below.

3.1.3 Research Questions

Referring to the objectives mentioned earlier, the research questions addressed in this thesis are formulated as follows:

- What are the determinants of software quality in global software work?
- What practices can be suggested to deal with the challenges of achieving quality in such contexts?

The following section describes the approach used to provide the answer to those research questions.

3.2 Interpretive Research

In this section, a brief discussion on interpretive research is presented along with the way it guides this study.

3.2.1 What is an Interpretive Research?

First of all, this approach was selected because, as mentioned in 3.1.1, the focus of this study was to understand a reality in which social and technological aspects were closely entangled. I share with Walsham (1993) that interpretivism is an *epistemological position* which is concerned with approaches to the understanding of reality and asserting that all such knowledge is necessarily a social construction and thus subjective [Walsham, 1993] p.5. Our knowledge of reality is gained only through social constructions such as a language, consciousness, shared meanings, documents, tools and other artifacts. Interpretive approach does not predefine dependent and independent variables, but focuses on the complexity of human sense making as the situation emerges. It attempts to understand phenomena through the meanings that people assign to them. This research is inspired by the principles for interpretive field research as formulated by the two authors [Klein and Myers, 1999].

Moreover, they argue that interpretive research can help IS researchers to understand human thought and action in social and organizational contexts. This approach has the potential to produce deep insights into information systems phenomena including the management of information systems and information systems development.

3.2.2 Conducting Interpretive Research

Klein & Myers (1999) propose a set of principles for the evaluation of interpretive field research in information systems, which are derived from the practice of anthropological research and the underlying philosophy of phenomenology and hermeneutics. The principles are not like bureaucratic rules of conduct, because the application of one or more of them still requires considerable creative thought. The authors do not suggest that those principles are mandatory, eventhough they do not advocate arbitrary selection either since the principles are, to some extent, interdependent. In this thesis, those principles were regarded as *an underlying source of insights* in conducting the research in various stages, for example: the principle of contextualization inspired me to write chapter 4 which described the contextual background of the case studies; the principle of interaction between the researchers and the subjects motivated me to reflect on how I interacted with the people in research settings, which also influenced the way I conducted the research; etc. Those principles were not used as a step-by-step prescription or a check-list, and therefore, their application was implicit throughout the study.

The principles are: (1) *The Fundamental Principle of the Hermeneutic Circle* which suggests that all human understanding is achieved by iterating between considering the interdependent meaning of parts and the whole that they form. This principle of human understanding is fundamental to all the other principles; (2) *The Principle of Contextualization* which requires critical reflection of the social and historical background of the research setting, so that the intended audience can see how the current situation under investigation emerged; (3) *The Principle of Interaction Between the Researchers and the Subjects* which requires critical reflection on how the research materials (or data) were socially constructed through the interaction between the researchers and participants; (4) *The Principle of Abstraction and Generalization* which requires relating the idiographic details revealed by the data interpretation through the application of principles one and two to theoretical, general concepts that describe the nature of human understanding and social action; (5) *The Principle of Dialogical Reasoning* which requires sensitivity to possible contradictions between the theoretical preconceptions guiding the research design and actual findings (the story which the data tell) with subsequent cycles of revision; (6) *The Principle of Multiple Interpretations* which requires sensitivity to possible differences in interpretations among the participants as are typically expressed in multiple narratives or stories of the same sequence of events under study; (7) *The Principle of Suspicion* which requires sensitivity to pos-

sible biases and systematic distortions in the narratives collected from the participants.

In order to construct an understanding of the data which mainly consists of software documentation and some observations and interview notes, I developed a framework which is elaborated in 2.5. This framework is mainly based on the three dimensions of software. It is important to get a more comprehensive picture of software development process with all its actors and factors. This framework is used to map the data, dissect and search for insights which may lead to the answer of the research problems.

In the next section, the empirical research setting is described along with the research activities.

3.3 Empirical Setting

This section describes the empirical research setting. Two case-studies have been conducted in an outsourcing supplier in India, **IndiSoft**[1]. IndiSoft is a software company offering software solutions for the worldwide market. The setting is selected for several reasons. Generally, India is leading in software outsourcing business, and is therefore the best place to learn about it. Specifically, IndiSoft has won an award for best practice in software development in 2002. Besides, the selected organization is medium sized and has clear focus on the software production instead of other outsourcing practices, such as call-centers, etc. For practical reason, the scope of this study can therefore be narrowed down to the capacity of the organization to handle projects. Access to the field site was gained through my supervisor, and was fortunate that there were no significant problems in gaining as well as maintaining access to the field site.

Initially, I intended to get a complete picture of how software was developed in all stages of its development by being with the development team and conducting an in-depth case study of a project from its beginning until its end. However, due to inevitable technical constraints, it was not possible to gain such access. Different ways of collecting data from previous projects were conducted instead. The data collected in this setting were in the forms of:

- Interviews with managers and developers mainly on the general operations in the company,

[1] a pseudo name

- Observation notes of several sit-in sessions of individual development tasks, project meetings and project management reviews meetings, and
- Documentation of two completed projects structured in a repository system, including Software Requirement Specification, Design documents, emails, chat transcripts, etc

The complete research activities are displayed chronologically in figure 3.3 below. In the next section, a brief introduction of the software industry in India is presented.

Now, the research experience is presented in the following section.

3.4 Research Experience

In this section, the research activities are described. As a beginner researcher, I found it interesting – at times, a bit frustrating – to learn that the research experience is not necessarily going as planned and how one should deal with such reality.

I ideally imagined that I would be able to be "in" an outsourcing project. Being "in" here meant that I would be a silent observer, or Walsham's outside observer [Imsland, 2003], following the team since the incoming of the project request, team building, and until the completion of software development life cycle. I planned to see how decisions would be made in the team, among the developers and along with the customers. I wanted to see the complete dynamics of the team.

I did not rely primarily on the use of interviews alone because in the software projects there was much more happening that could not be expressed verbally by the actors. I assumed that programmers would be good at what they did, but probably less able to express verbally what they were doing. It might be just such a too careful approach, but I wished, a bit ambitiously, to get the "whole picture". This was probably also because of the naïve tendency for something "concrete" as I had mentioned already.

The ideal scenario turned out to be impossible to realize. At that time, I didn't have any idea of how conducting fieldwork would be like, what strategy I should take in case the ideal scenario could not be done, and my inadequate skills in expressing what I wanted to do to the organization where I conducted fieldwork. This anecdote illustrates the reason. Several times I was assigned to some managers who were instructed by their superior to tell me the software processes used in the organization. I felt I had stated my intention clearly enough: to observe the software processes

	Activities	**Status/Methods**
Week 1 (25-30 Oct)	- Orientation - Introduction to practice groups in IndoSoft (SmartView & SmartBiz) - General data gathering (manuals & procedures)	Completed - Recorded Conversation - Soft Copy of manuals, project profiles
Week 2 (1-6 Nov)	Observation of on-going projects. Activities observed:	
	- Individual activities: o Sanjeev (a pseudoname), one project, construction phase, 4 sessions (@1.5-2.5 hrs) o Rana (a pseudoname), several projects, testing phase, 2 sessions (@2-3 hrs)	Completed - Researcher sits with the team member and observes the activities: on and off the screen - Note taking and recorded conversation
	-Group meeting o A 15-minute meeting of 3 team members, one project, one session	Completed - Researcher sits in the meeting - Note taking and recorded conversation
	- PMR-75 meeting, one session, 4+ hrs	- note taking
Week 3 (8-10 Nov)	Project Documents Collection: - access to VSS - collect non-business related information (as provided in the VSS – supposedly not the complete version) - projects: VMS and Point of Sale	Completed - As permitted, researcher opens each document and saves it to storing device. - Data is to be erased after the completion of the thesis writing
Week 4 (16-18 Nov)	Wrapping up Bid Farewell	

Figure 3.3: Research Activities

in an on-going project. However, none of these managers as well as the superiors had read my written proposal. They were informed about my

research by somebody else in the company orally and also by me when we were introduced. So, when I interviewed them, they told me what should be done instead of showing it; and showed me what was written in the manual instead of what happened in a particular project.

I never got to observe a project in the intended *ideal* manner. There were no newly-started projects waiting for me. Even if there was, it would take longer time than I had spared for the fieldwork. So, I learned, instead, how to interact with the organization. Several interviews took place with the managers and developers. It served mostly as a way for me to get familiarized with the processes in the company. I observed individual activities, such as coding and testing from several different projects; and collective activities, such as team meetings and Project Management Review. I came everyday to the office, stayed around the "production" area, did some observations and interviews, had lunch with them, etc. I knew from my reading that culture is an important factor in outsourcing phenomenon. By immersing myself with the life in the company I, naïvely, hoped to get a glance of the "cultural things".

Again, because one of the research goals was to understand more about GSW with the focus on issues of quality, I selected the interpretative research approach which guided me both as a way to gather and interpret the data. Interpretive studies assume that people create and associate their own subjective and inter-subjective meanings as they interact with the world around them. Interpretive researchers thus attempt to understand phenomena through accessing the meanings participants assign to them (Orlikowski and Baroudi 1991). Interpretive methods of research start from the position that our knowledge of reality, including the domain of human action, is a social construction by human actors and that this applies equally to researchers. Thus there is no objective reality which can be discovered by researchers and replicated by others, in contrast to the assumptions of positivist science (Walsham 1993).

The data is viewed as my own constructions of other people's constructions of what they are up to. This refers to Geertz's (1973) way of looking at interpretive data as our own constructions of other people's constructions of what they and their compatriots are up to [Geertz, 1973]. Social processes are not captured in hypothetical deductions, covariances and degrees of freedom. Instead, understanding social process involves getting inside the world of those generating it (Orlikowski and Baroudi 1991).

This chapter is concluded with a reflection on the challenges and limitations of the research are discussed in the following section.

3.5 Challenges & Limitations

In this section, a reflection on the research activities is presented. Some difficulties in organizing and understanding the data are also discussed.

3.5.1 Limitations of data

This research is primarily based on the analysis of documentations of post-mortem projects. Later on it was realised that there seems to be a mismatch between my intention – which is to get the whole picture of the project and all its aspects, including the subtle, non-technical ones – and the form of data available. On the one hand, the data can be considered complete in representing the whole development process, but on the other hand, it omits many lively aspects of the projects.

Reflecting on to the three software dimensions, it is now fully realized that the project documentation is actually more focused on the process dimension, as process is inevitably the most plausible thing to document.

Achieving the objective of getting the complete picutre of the project is not an easy task because the data was not structured for this purpose. The documents are technical and written while the development process taking place; in most cases, the authors are the developers themselves. So, it is natural that in the pace of work they are experiencing, there is not enough time to ponder the historicity of the documents later on. For example, minutes from chat sessions and meetings are recorded as it is. Only some of the information contained in the minutes, mainly about the software requirement, are extracted and summarised in a separate document.

I find many interesting occurences in the project by reading the actual chat session or meeting minutes. This gives flesh to the skeleton of formal, bureaucratic project documents. Combining the hard and soft aspects of the project shows how human being and technicalities entangle. My biggest problem in trying to do so was to determine where to start. So, I decided to use chronology, ordering everything in a timeline, as I believe it is the most resonant way for human mind. It enabled me to create a narrative out of the technical structures.

The next challenge was the volume of the documents. Technical documents usually follows a certain standards or templates which contains many repetitive and irrelevant details for my cause. However, reading them plainly is tedious and one may easily discard information as irrelevant too soon, especially when one has not determined which specific aspects to look for. Taking plain notes does not help much either because the documents were mostly in a summarized form. Doing so simply ends

up in another set of condensed documents. Reflecting on that, I decided to organize the data in the way that is described in the next section.

3.5.2 Data Analysis

I decided to use the Mind Map [Buzan, 1995] concept for note takings, as it represents a powerful graphic technique which provides a universal key to unlock the potential of the brain. It harnesses the full range of cortical skills – word, image, number, logic, rhythm, colour and spatial awareness – in a single, uniquely powerful manner. In so doing, it gives you the freedom to roam the infinite expanses of your brain. The Mind Map can be applied to every aspect of life where improved learning and clearer thinking will enhance human performance [Buzan, 2004].

I then started to ”mind map” the ideas contained in the project documentation by structuring them in units of time: the week. To make it possible to create large mind maps in an easier way, I used a computer-based mind map tool. The best start, I decided, was to map the activities recorded in the weekly status report to flesh out each week branch in the map. It was then followed by getting into details for each project documents that were created, reviewed or approved by a certain point of time. The software tool provided the facility to add long notes for any branch, so I had the freedom to retain the actual text or make keywords out of it.

The strength of this mapping was the ability to lay out the whole problem in one single screen. In contrast with the paper-based mind mapping, the software tools provided much flexibility which made this work less painful, such us expanding or closing up a branch like the tree-structure of the Windows explorer’s folders. One could create several maps to try different structures which might reveal subtle aspects of the projects. I would certainly recommend this way for future researchers who choose to dig up the documentation of old projects.

Chapter 4

The Context

This chapter presents the context in which the case studies took place. The organizational context is as important as the cases themselves because it is the scope in which QMS operates. The information in this chapter is mainly collected from the company presentations (powerpoint documents and websites), observations and interviews. This chapter begins with section 4.1 which briefly describes the larger context: the software industry in India. Section 4.2 then introduces the organization, followed by the use of *ecosystem* metaphor to describe its software development environment in section 4.3. Section 4.4 introduces the development cycle utilized in this organization. QMS elements in the organization is presented in the following section 4.5.

4.1 Software Industry in India

In this section, a discussion on Indian software context is presented with a focus on its emphasis on the pursuit of quality certification. Next, recent trends of software development practices in the country are also shown.

4.1.1 Why India?

Leading consultancy firms such as Giga, Forrester Research and McKinsey & Co. have cited various reasons for the increase of offshore outsourcing by Multi National Companies (MNC) to India. India's *quality* and *cost benefit edge* is one of the major draws for these organizations. Giga predicts that, compared to other competing countries such as China, Ireland, Israel, and the Philippines, India will continue to dominate as the preferred offshore country. According to a study conducted by Forrester in November,

2001, India's edge over other competing nations in the IT outsourcing business is based on the country's decade old experience in this area, fluency in the English language, supportive government policy infrastructure, and high quality offerings.

Today, MNCs are rushing into India to stake a claim to the IT outsourcing market. While a large number of companies are outsourcing their software development to Indian companies, others are establishing a presence in India and participating actively in the software export game. The MNC sector emerged as an important segment contributing Rs 9,855 *crore* (USD 2,188 mill) of the total exports of Rs 35,600 *crore* (USD 7,905 mill) in the year 2001-02, translating into 27% share of the total exports. (Source: NASSCOM [1] – India's National Association of Software and Service Companies)

4.1.2 The Pursuit of Quality

According to NASSCOM, in the past few years, the Indian IT industry has pursued the goal of attaining the highest international standards of quality. A World Bank funded study conducted as early as 1992 to discuss Indian software strategies had concluded that more and more vendors in the US prefer to get their software development work undertaken in India for its quality and cost advantage. I argue that quality certification has become both a material and symbolic value to create trust between clients and vendors.

Indian players have created a strong value proposition in the IT software and services arena. India enjoys advantages of people sophistication in terms of a very large pool of English speaking scientific personnel, varied and extensive skill sets in terms of technology, and offering services at globally competitive costs. India also boasts of vendor sophistication– with more than 200 companies being quality accredited and serving the needs of over 255 of the Fortune 500 companies. Today, the world looks towards the Indian IT software and services industry for its good quality and high price performance. According to McKinsey & Co., India has and will continue to have a growing number of vendors successfully working on complex projects across all areas of software and services, and performing at levels comparable to those of leading global players.

As of 31st March 2002, India had 42 companies at SEI CMM Level 5 assessment. The quality maturity of the Indian software industry can be measured from the fact that already 316 Indian software companies have

[1] `http://www.nasscom.org/artdisplay.asp?cat_id=28` Accessed: 22/09/05

acquired quality certifications and more companies are in the pipeline to do so.

In March 2002, NASSCOM conducted a survey to ascertain the adoption of international quality standards by IT software and services companies in India. An analysis of top 300 companies in India revealed in figures 4.1 and 4.2 below [2].

Quality Certification	*No. of Companies*
Already acquired ISO 9000 or SEI and other certification	216
Companies acquired quality certification by March 2002	6
Certification expected between April-December 2002	64
Additional certification expected during 2002	70
No plans at present	22

Figure 4.1: Quality Certification

SEI Quality Assessment	*No. of Companies as on 31st May, 2002*	*No. of Companies by 31st March, 2003*
SEI CMMi	3	5
SEI CMM Level 5	42	46
SEI CMM Level 4	22	38
SEI CMM Level 3	24	34
SEI CMM Level 2	3	16
PCMM Level 5	2	2
PCMM Level 4	1	2
PCMM Level 3	5	6
PCMM Level 2	4	12
Note: Some companies have multiple certifications.		

Figure 4.2: SEI Quality Assessment

The other heartening feature of the industry has been the growing acceptance and adoption of the newly emerging People-Capability Maturity Model (People-CMM) by the Indian software industry. For a country like India, with its large assets in the form of skilled human resources, the relevance of People CMM needs no emphasis. A large number of Indian IT software and services companies have been quick to realize this and

[2] `http://www.nasscom.org/artdisplay.asp?cat_id=208` Accessed: 22/09/05

have either implemented or initiated such certification programs. (Source: NASSCOM [3])

4.1.3 Software Practices in India

This sub section briefly presents some findings in literature which illustrate the recent trends of software development practices in India. Cusumano et al (2003) in their paper titled "Software development worldwide: the state of the practice" documents some of the Indian software practices and compares them with those of other places in the world such as Japan, US and Europe. The paper reports early descriptive results from a new global survey of completed software projects that attempts to shed light on international differences in the adoption of different development practices. The practices are illustrated in the figure 4.3 below [Cusumano et al., 2003].

The practices used

	India	Japan	US	Europe and other	Total
Number of projects	24	27	31	22	104
Architectural specifications (%)	83.3	70.4	54.8	72.7	69.2
Functional specifications (%)	95.8	92.6	74.2	81.8	85.6
Detailed designs (%)	100.0	85.2	32.3	68.2	69.2
Code generation (%)	62.5	40.7	51.6	54.5	51.9
Design reviews (%)	100.0	100.0	77.4	77.3	88.5
Code reviews (%)	95.8	74.1	71.0	81.8	79.8
Subcycles (%)	79.2	44.4	54.8	86.4	64.4
Beta testing (% ≥ 1)	66.7	66.7	77.4	81.8	73.1
Pair testing (%)	54.2	44.4	35.5	31.8	41.3
Pair programming (%)	58.3	22.2	35.5	27.2	35.3
Daily builds (%)					
At the start	16.7	22.2	35.5	9.1	22.1
In the middle	12.5	25.9	29.0	27.3	24.0
At the end	29.2	37.0	35.5	40.9	35.6
Regression testing on each build (%)	91.7	96.3	71.0	77.3	83.7

Figure 4.3: Software Practices in India and other countries

Most of the projects being studied (see fig. 4.3) use architectural, functional, and design specification documents rather than just writing code with minimal planning and documentation. These traditionally well-regarded practices are especially popular in India, Japan, and Europe. India use detailed design specifications very intensively, reported as 100 percent in one study. Indian projects to some extent tend to vary with the use of daily

[3] `http://www.nasscom.org/artdisplay.asp?cat_id=35` Accessed: 22/09/05

builds over a project's life. India has a relatively high concentration of running regression tests on builds in projects. The use of paired testers and paired programmer techniques are especially popular in India.

US programmers often have different objectives and development styles. They tend to emphasize shorter or more innovative programs and spend more time optimizing code, which ultimately reduces the number of lines of code (LOC) and increases effort. The Indian organizations might well have adopted the US-type programming style because they often have a significant proportion of US clients and, besides, many Indians developers are educated in the US, which makes it easy to import US-centered practices to India.

Cusumano et al (2003) conclude that there is some truth to the recent speculation that Indian software companies are increasingly adopting more formal quality management techniques. Finally, they say (p.33-34), "Our data suggests that anecdotal evidence emerging about the process and practice strengths developing in India are well founded. Above all, our data shows that Indian organizations are doing an admirable job of combining conventional best practices such as specification and review with more flexible techniques that should let them respond more effectively to customer demands. If the broader population can replicate this trend, the Indian software industry will likely experience continued growth and success [Cusumano et al., 2003]."

4.2 The company: IndiSoft

According to the company profile document, IndiSoft was established in one of the major Indian IT centers to execute overseas software projects and provide on-site support to its customers world-wide. Marketing is carried through the strategic tie-ups and marketing associates based in the United States, Europe and South-east Asia. The organization's philosophy is to provide high quality software solutions and professional assistance to its clients in managing their Information Technology growth in an optimized and cost effective way.

IndiSoft offers a complete range of software services to its customers around the world. Some of the areas include: systems feasibility and requirements studies, design and development of both applications and systems software, conversion, migration and maintenance of software across different platforms and computer systems.

IndiSoft's professionals were admitted by the company to own a wide spectrum of skills. These professionals are recruited through rigorous test-

ing and selection processes. The organization augments its human resources base and skills spectrum at different levels from time to time. It places great emphasis on training and developing its professionals on a continuous basis to meet the skill demands of its projects and also to improve the quality and productivity of its services to its customers. To accomplish this, it has established a training group. The training programs are designed and implemented so as to ensure: proper induction of the new employees, compliance with the skill proficiency requirements of the project teams, and continuous updating of the skills in state-of-the art technologies and methodologies.

Stated in the Quality Manual – one of the major documents in the organization from which software practices are drawn – the company is over 10 years old, medium sized of around 200 employees. It has delivered 3000 man-year worth of projects. The organization is structured as in figure 4.4 below.

The structure of the organization (see fig. 4.4) is, I believe, a *professional bureaucracy* – one of the five structures proposed by Mintzberg (1983). "The Professional Bureaucracy relies for coordination on the standardization of skills and its associated design parameter, training and indoctrination. It hires duly trained and indoctrinated specialists – professionals – for the operating core, and then gives them considerable control over their own work [Mintzberg, 1983] p.190."

Being a professional bureaucracy as described above, it is implied that IndiSoft professionals work relatively independently of their colleagues, but closely with the clients they serve. According to Mintzberg (1983), the structure of this organization is essentially bureaucratic; its coordination is achieved by design, by standards that predetermine what is to be done.

Yet, the organizational structure is also simple and dynamic. It is simple because there are no layers between the CEO and the functional units which share equal position in terms of coordination (see fig. 4.4). The use of technology and the setting of the workplace, as I observed, also enables more dynamic communication because it accomodates the current ongoing projects. Some functions are always available, but the volume of the work and the relations between the units are mainly affected by the current projects.

This dependency on the projects makes the organization classify the projects through several criteria, set up several divisions – which they call *Practice Groups* – and develop standard procedures for each project classification. As written in the company profile, the divisions are called: SmartView, SmartBiz and System & Networking. One manager explained that in general, the division of projects was made based on the time and

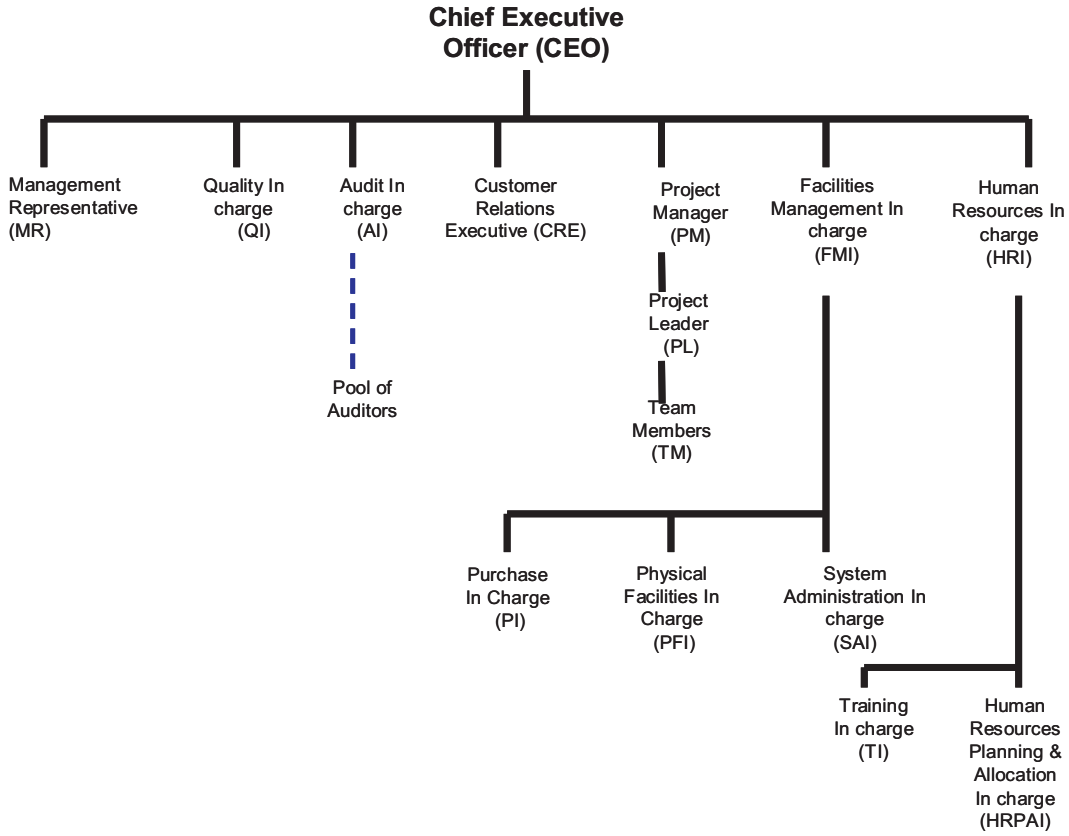

Figure 4.4: The Structure of the Organization (Source: Quality Manual)

the efforts required (long or short term) and the type of work (development or maintenance). SmartView handles a relatively short term (about 1000 hours) development projects, SmartBiz takes care of long term development projects which often require functional and domain expertise, and System & Networking handles maintenance jobs, such as networking, porting, etc.

4.3 The Ecosystem

In this section, the context is examined more closely using the *ecosystem* metaphor which suggests that every element in it affects everything else. Cockburn (2001) suggests the term ecosystem to describe the software project team and its context. A software project sets up a small ecosys-

tem made up of personalities from diverse cultures. The ecosystem may include: walls acting as barriers and open spaces as conduits, people in their professional specialties acting as interacting subspecies, individuals with strong personalities changing the way in which the ecosystem works. He emphasizes that everything affects everything: the chairs, the seating, the shape of the building, whether people share a native language, or even the air conditioning.

I am going to expand the term ecosystem to the scope of the organization based on my observation in the field. I attended the office almost regularly, like the other employees in the company, for four weeks. Some employees referred to me as a trainee. I was given a table with a workstation where I could get access to standard facilities such as word processing, internet, IP messenger, and the VSS system.

The company is located in a five-storey building which has two wings. IndiSoft occupies the ground floor and the top floor. The rest of the place in the building is occupied by other companies. The top floor is the office for the Business Development (BD) unit. Some managers and BD staff have their offices or tables on this floor. My workstation was located in the BD staff office. As I observed, the production takes place on the ground floor, on both wings. SmartBiz and SmartView share a wing and the other one is for System & Networking. CEO and Group manager's office are also located in the ground floor. The setting suggests the closeness of the highest power in management to the production units – just as hinted by the organization structure (fig 4.4). Their presence is always apparent to the software developers.

Internal communication system is established for effective implementation of the QMS. As written in the manuals, the quality procedure includes communications pertaining to: project status, performance of QMS, meeting customer requirements, customer satisfaction levels, management directives and decisions so as to ensure effective implementation of QMS. Project Managers ensure that the related processes and plans are communicated to all personnel involved in the projects and others who are expected to follow the same. The internal communication across the organization occurs through media like emails, e-portal, notice boards, circulars, status reporting (weekly/monthly), review meetings, management reviews, intra-project meetings, open discussion with PL's and PM's of different projects and management (Source: Quality Manual).

I noticed that the development room was divided by cubicles occupying the middle space and surrounded by meeting/training rooms, managers' room, etc. Each cubicle space is shared by three to four programmers with their own machine. Some of the cubicles are equipped with

telephones. Each workstation has an IP messenger connecting everybody logged on the system.

During the observations, I noted that everybody seemed to be engaged with their own assignments. Some of them, usually sharing two facing cubicles containing 3 - 6 people, seemed to work together. It was often the case with SmartBiz projects. Sometimes team meeting was scheduled and took place in one of those well facilitated meeting rooms available. The movements of people from one cubicle to another, discussing a piece of code or clarification of testing activity, happened quite often without getting too noisy. Often, one developer called another via IP messenger, asking for something, and if it could not be settled online, one would just visit the other's cubicle. However, the seating arrangement did not reflect the grouping of people based on the same project/task. One person whom I observed was engaged in two different tasks from different projects in one day. She was doing the testing for a project in the first half of the day, and in the other half of the day, she was writing a use case for another project in the early phase with someone who seemed to be junior in the company.

People started working at 10.00 and finished quite late in the evening, like after 18.00. However, sometimes, I found one developer, whom I observed, was still online on the messenger very late till 22.00. I asked him and found out that he was working and, now and then, he communicated online with the client from the Middle East. The company seemed to tolerate and be flexible with the time difference with the customer. This particular developer had, on some occasions, to work on Sunday, because the client's "weekend" was on Fridays. In the company, people worked from Monday to Friday plus, every second Saturday.

On one Saturday, I had the opportunity to sit in a Project Review Meeting. It started at 14.00 and continued until over 18.00. This meeting was attended by all managers. The Head of Quality Department facilitated the meeting by being the critic to the projects presented by the Project Managers, while other managers observed and commented. The meeting started with project presentations. The status of the project was examined; risks and issues were addressed. While a project was being presented, usually its team members and project leader were also present to help the project manager answer the questions from the critics. I observe that this meeting was very dynamic and it seemed important. People came and went; those who had finished with their presentation might just leave and continued with their work. Only the facilitators were present during the whole meeting. Following the project presentations, there were presentations of other non project-related divisions, such as Human Resources,

Training Division, etc. In the meeting, people seemed to be open and, sometimes, critical to the presentation in a constructive way. Often, the CEO, who was sitting next to me, spotted other related issues from the presentation which concerned the management in general; he, then, spoke it out to the floor. In general, I felt the atmosphere was close and relaxed, and not hierarchical as I had assumed it would be.

When a project was offered by a client, the BD unit would usually catch it first. They would then discuss it together with the technical people, usually the project manager from one of the practice groups. I observed, indirectly while I was working in the same room on the 5th floor, that the project manager would come upstairs and share his/her technical perspectives with the business oriented people. When the project proved feasible, both technically and financially, then the project could be initiated and – I playfully imagined – "dropped down from heaven" to the ground floor for production.

4.4 Development Life Cycle in the Organization

This section presents the standard development life cycle in the organization as written in two manuals: Project Procedures and Software Project Process, shown in fig 4.5 below. The organization calls their standard processes as *Development Life Cycle (DLC)* which is a variant of SDLC (see 2.3.2). IndiSoft modifies the standard by including prototyping in the requirement analysis phase. In the two case studies, I noticed that the DLC term was stated in the project plan to indicate methodology used for those projects. DLC generally consists of nine phases, namely:

1. *Start up*, in which project plan preparation and environment setup take place.

2. *Requirement Analysis*, in which detail plan preparation, study of existing system and documents, prototyping and SRS preparation take place.

3. *High Level Design*, in which decomposition of requirements, definition of system architecture, database/file system design, process /data /object modeling, and prototype refinement take place.

4. *Low Level Design*, in which identification of low level components, program/report/screen specification, interface definition, and test plan preparation take place.

5. *Software Construction*, in which construction of the program code takes place.
6. *Integration and Testing*, in which the integration of unit tested group takes place.
7. *Package & Release*, in which installation of the product takes place.
8. *Acceptance Testing*, in which the team provides customer support.
9. *Wind Up*, in which release of resources and completion report take place.

In the manuals, each phase and its corresponding activities are defined in terms of their entry criteria, tasks, verification and validation procedures, exit criteria and work products. Planning is an activity found in every phase. In the startup phase, it is in the form of a project plan, while in all other phases, it is the detailed plan for the phase. Each phase consists of different number of tasks; the largest is the *design* phase with 8 activities, while the shortest is *acceptance testing* with only 3 activities. The key roles for each activity are also defined. It is usually either the client or IndiSoft or both. More than half of the activities require close relation between the two. Only one activity is affected by the client alone, namely: User Requirement Specification. There are three activities that span along the whole phases, namely: Quality Management, Code Walkthrough, Testing, Project Management, Review Meetings, Change & Configuration Management, and Change Control.

4.5 QMS in the Organization

This section presents the QMS used in the organization. According to the *Quality Manual*, QMS of the organization is defined and documented in terms of Quality Policy, Quality objectives, Quality Manual and the associated Procedures Manuals. The documentation is organized as shown in fig 4.5 below. The information presented in this section is collected and extracted from the manuals.

Quality Manual constitutes the apex manual (see fig. 4.5). It lays down the quality policy, quality objectives and directives for procedures to be evolved as addressed by the latest ISO 9001 standards. *Software Project Process (SPP) Manual* defines the sequence and interactions of the processes to be carried out for the projects or assignments undertaken by the organization within the scope of the Quality System. *Project Procedures (PP) Manual*

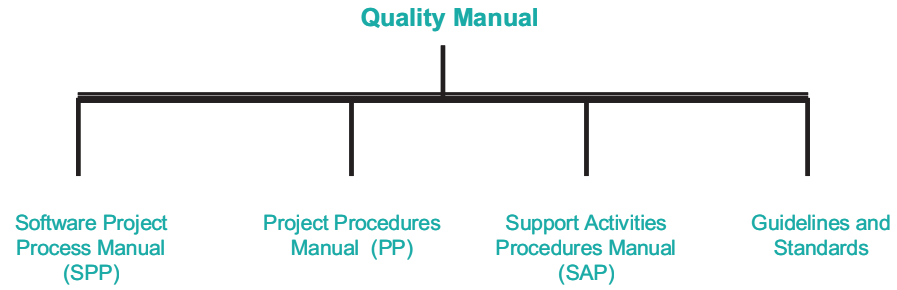

Figure 4.5: Quality Manuals

covers the procedures to be followed while executing the activities related to the processes defined in the SPP. *Support Activities Procedures (SAP) Manual* covers all the procedures applicable to the support functions namely, Quality Assurance Group, Audit Group, Human Resource Planning and Allocation, Training, Purchase, and Infrastructure facilities. *Guidelines and Standards* is about the guidelines and standards applicable across all activities. IEEE standards are adopted as guidelines for most of the activities.

4.5.1 Organizational Structures

As previously mentioned, the organizational structure (fig. 4.4) along with the quality responsibility assignment is defined in the Quality Manual. This matter has been discussed in section 4.2.

4.5.2 Responsibilities

Quality Manuals determines roles that are responsible and authorized in the implementation of QMS. The roles are displayed in the organizational structure (fig. 4.4). I would elaborate only those which are closely related to the implementation of QMS in the project, including: Project Manager, Project Leader, and Team Member.

A **Project Manager (PM)** is nominated by and reports to the CEO. PM heads a set of projects and/or lines of business as assigned by the CEO. Meeting overall project commitments and customer satisfaction is his/her responsibility. PM manages project teams, project executions and customer interactions. His/her responsibility includes among others: (1) planning and initiating process for acquisition of resource and skill requirements for projects; (2) suggesting methods, tools, methodologies, and

techniques to improve the process in consultation with the client when needed; (3) ensuring effective implementation of quality system in the projects; (4) reviewing the training requirements; (5) reporting status and issues to the management; (6) initiating and participating in the joint reviews with the clients; (7) handling non-conformities in the project; (8) formal closure of the project; and (9) participating in the performance appraisals of the personnel involved in the project. PM is authorized to, among other things, coordinate with the customer, assign and delegate responsibility and authority to the Project Leader, review and approve detail plan of the project, approve changes in the requirements, approve project related documents and deliverables, conduct progress review with the Project Leader, and maintain required level of customer liaison for the projects.

A **Project Leader (PL)** is nominated by HR Planning & Allocation In charge (HRPAI). PL plans and executes the project according to the plan and ensures the required product quality and process compliance is maintained. In doing so, PL is supported by the project team. He/she gives the required technical support to the team. His/her responsibilities, among other things, include: (1) planning the project in consultation with the PM; (2) ensuring that the training requirements are identified and given; (3) implementing the quality system in the project; (4) maintaining the configuration management in the project; (5) interacting with the team continuously and conducting the team meetings as planned; (6) ensuring the use of tools and techniques in the projects; (7) analysing the cause for non-conformities and taking the required corrective and preventive actions; (8) reviewing the project related documents and work done by the team; (9) reporting to PM, client and management on the status of the project; (10) collecting and analysing the planned metrics for the project and reporting to the management; (11) participating in reviews and audits, and personnel performance appraisals; and (12) winding up the projects and handing over the records and documents to PM for further action.

A **Team Member (TM)** is allocated by HRPAI and reports to PL. TM carries out all the activities assigned throughout the projects life cycle, including: analysis, design, construction, testing and validation, and any other tasks assigned as per plan and reporting their status. His/her responsibilities are to: involve in the QA activities, comply with the defined quality system in all their activities, adhere to standards, rules, practices and conventions, generate the project quality records that are complete and correct, participate in team meetings, and take timely and appropriate corrective actions for non-conformities found in the work products and the area of activities. TM is authorized to use suitable tools, methodolo-

gies, and techniques identified for the projects.

4.5.3 Processes and Resources

In the Quality Manual, the *customer related processes* are defined, including determination and review of product related requirements, and customer communication. The product requirements are analyzed from the customer supplied user requirements document and through customer interactions. These requirements include: functional and operational requirements specified by the customer, requirements not specified by the customer but necessary for intended or specified use, statutory and regulatory requirements related to the product, and any additional requirements determined by the organization. If user requirements document is not supplied, the project team will produce one, in consultation with the customer, to the required level of detail int he form of project proposal.

The *user requirements* document is reviewed internally before seeking the customer's confirmation. The review process ensures that the requirements are clear, issues identified earlier are resolved and the organization has the capability to meet the contract obligations. The amendments to the requirements are documented and are subject to the documented change control procedure. Adequate control on making the relevant changes in the project documentation, on account of changes in the requirements, and informing all concerned project personnel is ensured through documented procedures.

Communication with customer is conducted by the organization at various levels during the product realization process. Prior to project initiation – i.e. during the stage of handling customer inquiries, proposal making, negotiations, contract finalization – communication is conducted by a senior level person from the marketing group, authorized by the management for this purpose. The responses to all tenders/ proposals and contracts are, however, maintained at the organization level, with the CEO or the designated person by CEO. During the project lifecycle, the project manager is handling the customer communications on project specific matters with the designated project in charge on the customer side. This includes matters relating to product specifications, schedules, progress reports, etc.

Joint reviews are arranged at least once every month to ensure faster resolution of the issues, and to ensure that there is common understanding between the customer and the team executing the project. Record of evidence of all such communication will be available with the projects.

Any changes to the agreed upon requirements leading to changes to the contract and billing are handled in close coordination with business development group. The communications is handled by the project manager, by default, or a designated person from the marketing group, as decided by CEO. The decision is made to ensure the effectiveness in maintaining customer satisfaction levels and sensitivities and for the best interest of project execution activities. It will be ensured that both the project manager and the marketing person are in accordance to all amendments to the contracts.

The Quality Manual also defines the following processes concerning design and development, such as, planning, inputs, outputs, review, verification and control.

A detailed *project plan* covers the design and development process, which is reviewed and approved before initiating the process and updated as the process progresses. It includes: all the major activities that form part of this process and their sequence; review, verification and validation tasks relevant to this process; roles, responsibilities and authorities for the design and development process; support and resource requirements internally and on customer side; customer interaction requirements; metrics mapping to the quality objectives of the project and their collection.

The *Requirements Specification* is the input for design and development process which is provided by the customer or prepared by IndiSoft in consultation with the customer. For the latter, the requirement specification is reviewed internally before it is submitted to the customer for formal approval. The Requirements Specification is written based on the user requirements and it forms the basis for all the phases in the life cycle of the product. It is written in an exhaustive manner and will include applicable information derived from previous similar product, if any, such that it should enable error free design and development. Adequate controls is maintained to track the traceability between the user requirements accepted in the contract, quality objectives for the project, detailed requirement specifications prepared during this process and the subsequent stages in the product realization stages. The Requirement Specification also covers the following features as applicable to the project: safety requirements, user interface requirements, report layouts and contents, performance requirements, reliability features, security and privacy features, and regulatory and statutory requirements. All ambiguous or conflicting requirements shall be resolved mutually with the client.

Design process follows the design consideration, associated design rules, and applicable past experience. The project team selects an appropriate systematic design methodology based on the project and contractual re-

quirements, which include testing and maintenance facilities. The design phase output is documented in a design document in such a way that each of the design elements is traceable to the requirement specification. The design review ensures the conformance of the product with the defined requirements and any deficiencies must be resolved before proceeding to the next stage. The customer, if required contractually, approves the design document before it was executed.

Next, the design document becomes the basic input for the development or construction activities. As you notice, the term construction and development are used interchangeably. The *construction* phase utilizes suitable methodologies, programming rules and languages, consistent naming and coding conventions and commentary rules, and follows any stipulations in the contract. It also uses appropriate verification and validation methods, and test plans identified in the Project Plan. The construction process is reviewed and deficiencies found must be resolved before continuing to the next stage.

4.5.4 Procedures

The procedures for activities carried out throughout the execution of projects or for support functions in the organization are defined in the Project Procedures Manual. The activities included in this manual are contract/proposal review, project initiation, team building, defect handling, obtaining customer sign off, change control, configuration control, code walk through, software unit testing, preparation of user manual, system testing, final inspection, etc. The activities are described in terms of its purpose, scope, entry criteria, responsibilities, procedures, verification and validation, exit criteria and work products.

4.5.5 Measurements

The Quality Manual specifies measurements which ensures, as literally written in the document, the product compliance and the process compliance. All planned activities are monitored closely, while the product as well as the process data is collected on a continuous basis, and reported to the management periodically. This is done by using metrics which include data reflecting the levels achieved of quality objectives and the customer satisfaction. The metrics are analyzed at the project level as well as at the organizational level. Project planning is supposed to identify the needs for statistical tools and techniques for recording and analysis of measurement

data.

Customer perception is monitored continuously from the regular communication, joint reviews, conference call minutes, etc. A role, Customer Relations Executive (CRE), is dedicated to seek inputs from the customer on regular basis, at least once every quarter on customer satisfaction levels throughout the project. At the end of the project, formal feedback on the project and the product is obtained objectively through surveys. CRE is continuously informed by all projects about the customer complaints or satisfaction reports/mails, while customer is informed by the status report.

Internal Quality Audits, conducted by trained auditors, are performed quarterly or earlier in such a way that every project and/or support group is audited at least once every half year through planned arrangements. It is also ensured that every project is audited at least once during its life cycle. Internal Quality Audits is scheduled on the basis of status and importance of the activity. The result is documented and brought to attention to the personnel responsible for the audited area and timely corrective actions are taken.

The Quality Manuals also specifies, separately, the monitoring and measurement of process and product. *Process* monitoring and measurement is conducted by Quality In-charge (QI) who ensures that: QMS is available to all personnel, required training is provided, and necessary support for implementation on request is required. Furthermore, all projects and function of the organization must ensure that documented system is implemented and conduct periodic self-monitoring to ensure process compliance.

Product monitoring and measurement is described as the followings. All the defects reported by the peer and at all other levels within the project, by the QA group, and by the customer, formally or informally, are logged in the defect logs and monitored closely. The data collected in the defect logs will be analyzed regularly and reported to the management giving the product compliance levels. The projects are responsible to monitor the product compliance levels and strive to continuously improve upon the same.

Chapter 5

Case Study 1: Versatile Messaging System II

This chapter presents the first case study: Versatile Messaging System II. It starts with the project description in section 5.1, followed by the chronology showing the week-by-week progress of the project in section 5.2, which is extracted from the weekly status report. Furthermore, the quality framework is applied here to analyse the case in section 5.3.

5.1 Project Description

VMS-2 was a multi-tiered web-based application for communication, transaction and information portal developed for a US-based client, HealthCo[1]. This application was to be used in a medical environment, such as a hospital and a physician's office, as a facility for the hospital staff, physicians and non-physicians staff to send a page, text message to cell phone, email and fax to each other. This application should be fast and reliable since it was to be used in situations where time could be a very critical factor. Besides, the system should be able to log all the occurring communications for reports. The first version of the system had been developed previously by IndiSoft. The project in question was to develop the second version of it.

Briefly, the scope of the project was to develop the application in ASP .NET (no information was available about the environment in which the previous version was developed); change and add several functionalities; make general changes; and design a new graphical user interface.

[1] A pseudo name

According to the company presentation documents, which are usually given to potential customers, IndiSoft had successfully developed VMS such that it could provide the desired functionality such as: message passing using centralized web interface to the recipient. The features of VMS were, among others, messaging (page, SMS, email) system between hospitals and doctors, among doctors, and broadcasting to all registered users.

IndiSoft concluded, as written in the project presentation, that the resulting system had the desired quality including: increasing productivity and efficiency due to its less communication time required and lower operating cost, compatibility with existing assets, ease of use for all users, and support standards to ensure long-term returns. Quantitative data describing the quality of the work was also available in the project documentation, recorded in the metrics, such as: schedule slippage, review effectiveness, project rework, effort variance, compliance index and on-the-project training.

5.1.1 Project Management

This brief section illustrates the way the project was managed, including the use of documentation, methodology, technologies and tools. The project team consisted of three people: a project manager, a project leader and a team member. The project used, internally, several deliverables to document the processes such as: Software Requirement Specification (SRS), Software Design Document (SDD), Test Report, User Manual, Final Inspection, Code Review Reports, Project Delivery Note, and Project Completion Report.

The methodology or process model used in the project, as stated in the Project Plan, was Waterfall Model, which actually was the IndiSoft's own version of the standard SDLC: the DLC (see 4.4). The application was developed on Windows 2000 Professional platform, using MySQL and SQL Server 2000 database, ASP .NET language, and other tools such as Visual InterDev 6.0, MS FrontPage 4.02, IIS 5.0 and Visual Source Safe 6.0. It was developed by using four machines of P-III, 256MB RAM; 1 GB HDD. The Visual Source Safe 6.0 was used as a tool for version control and documents archive.

5.2 Project Chronology

This section presents the case chronologically, which is resurrected from the project documentation alone and with occasional references to the Project

Procedures and Software Project Process manuals. The project took place for around 15 weeks according to the available weekly status report. According to the project plan, the project was initiated in late August and was expected to finish in mid December 2003. However, the weekly status reports available in the archive were only until the third week of November. Throughout the project, the team kept records of the effort in hours and included them in the weekly status report so that the customer could keep track of the activities and the effort spent on the project. I noticed from the project documentation that there was a period (3 – 4 weeks) for user acceptance testing during which the weekly status report was no longer produced. According to one of the metrics, the total effort utilized in this project was 1111 hours. The different development activities were grouped into categories such as Project Management (PM), Product related (Pr), Quality Assurance (QA), Review & Meetings (RW), Training (Tr) and Others (O). The complete list of activities is presented in figures 5.1 and 5.2 below.

Start-up Phase

The project was started right in the third week of August 2003. IndiSoft and HealthCo – the US-based client – had agreed on several matters which made the project possible to execute, as recorded in the project documentation. The agreement was written in the formal contract that bound the two parties. As prescribed by the Project Procedures, a review had to be conducted to ensure that the scope was well-defined, the requirements were adequately defined and documented, and the responsibilities of the client and Indisoft were clearly defined, both client and IndiSoft possessed the capability and resources to meet the contractual requirements, and joint reviews and periodic progress reporting was established. Should there be any corrective actions, the head of the Marketing Group and/or Customer Relation Executive would take action. There was *no* written information about the contract review stage, however as with other projects, this project should follow the procedures prescribed in the manual as elaborated above.

The scope of the project, defined at this stage, involved enhancing the existing functionality of VMS-1 application developed by IndiSoft. The proposed enhancements are divided into following categories: *Paging* related module changes, *Physician on Call* scheduling related changes, *Master Data* entry related changes, general changes and migration of the application from current databases MySQL 3.55.23 to MS SQL 2000 and GUI redesigning.

Week	*Activities*	*Effort*
1	▪ Project initiation (completed) ▪ Requirement analysis (ongoing)	Total: 48 (PM)
2	▪ Project Plan preparation (completed) ▪ SRS preparation (ongoing)	Total: 84 (PM: 15, Pr: 64, QA: 5)
3	▪ SRS preparation: (ongoing) – after receiving feedback and clarification of the first draft from the client. ▪ Preparation of *layout* (completed)	Total: 79 (PM: 10, Pr: 69)
4	▪ SRS preparation (ongoing) with the second draft being written and sent for approval. ▪ *Layout* preparation (completed) with some suggestions for change from the client.	Total: 82 (Pr: 72, QA: 10)
5	▪ SRS is reviewed and approved by the client. ▪ SDD preparation (ongoing) ▪ The team also receives some suggestion for modification of the layout template from the client. ▪ Coding standard document has been written, code walk through checklist has been prepared and test plan preparation has been started.	Total: 67 (PM: 15, Pr: 40, QA: 8, Tr: 4)
6	▪ Design document preparation (ongoing) ▪ Database Design and Architecture (completed). ▪ The test plan (completed) and sent for review. ▪ The unit test cases are being prepared; so far they have finished with *VMS-2 admin* and *Physician Office admin* modules. ▪ Template design is in progress: *menu identification* part was completed. ▪ Coding standard definition (completed). ▪ Code walkthrough checklist has been reviewed and approved.	Total: 57 (PM: 4, Pr: 45, QA: 6, Tr: 2)
7	▪ The followings are completed: template design, unit test cases, pseudo code for complex functionality, test plan approval, and database design	Total: 57 (PM: 4, Pr: 45, QA: 6, Tr: 2)
8	▪ Consolidation of SDD (ongoing) ▪ Design changes related to the *Billing* module ▪ Complex queries are documented ▪ Test data are set. ▪ *Time Zone* approaches are finalized.	Total: 60 (Pr: 58 and QA: 1)
9	▪ Design document is approved by the client. ▪ Design changes related to *billing* module, complex queries documentation and test data setup, construction of common layout template, menu design and *list screen*. (completed)	Total: 171 (Pr: 159, QA: 3, O: 9)

Figure 5.1: VMS Chronology

Project Initiation Form was written and became the official statement signifying the approval and commitment of IndiSoft management to com-

10	▪ *List screen* with search functionality in all level (completed) ▪ *Add/Edit screen* for admin (completed)	Total: 128 (Pr: 127, QA: 1)
11	▪ The following modules: *Add/Edit functionalities for Physician Office* and *Hospital and Message groups including affiliation* (completed) ▪ These modules: *Create and View Schedule functionality* (in progress)	Total: 117 (Pr: 116, QA: 1)
12	▪ Modules *Add/Edit functionality for Physician and Non Physician at PO* and *Hospital* (completed) ▪ The team deploys the application for client review while still working on *Create Schedule* module	Total: 66 (PM: 5, Pr: 60, QA: 1)
13	▪ *User site* screen (completed) ▪ The application is deployed for client review	Total: 44 (PM: 3. Pr: 40, QA: 1)
14	▪ The testing phase is started. ▪ The team deploy the application for client review (as in the status report)	Total: 68 (PM: 3, Pr: 5, RW: 60)
Note: ▪ **Efforts are in hours.** ▪ **Effort categories: Project Management (PM), Product-related (Pr), Quality Assurance (QA), Review & Meeting (RW), Training (Tr), and Others (O)**		

Figure 5.2: VMS Chronology (ctd.)

mence the project execution and activate the required resources. This form was also the zero hour marking the point from which all the hourly effort spent in the project would be counted. As written in the procedures, the CEO would assign a Project Manager (PM) and introduced him to the customer as he would be in charge of maintaining customer satisfaction and comfort levels. The PM then would assign a Project Leader (PL) after consulting it with the CEO and the human resource executives. The PL then prepared a project profile. The Project Initiation form recorded the details of the project such as client and project description, expected start and end dates, estimated effort in person-months, and resource requirements.

The team was expanded by adding one Team Member to the already available PM and PL, forming a group of three members. The human resource requirement plan was normally written by the PL, reviewed by the PM and approved by the CEO. All team members would undergo a comprehensively planned project induction where all the necessary knowledge of project generals and specifics were given through briefing, self study material or training. There was no information whether the current team members were involved in the development of previous version of VMS; therefore I could not know how familiar the team members were with the project before they were assigned. However, the training log did

not register any induction training.

The Project Initiation form also mentioned the four-week User Acceptance Phase after the completion and testing of the system, as required by client, in which at least one person would be assigned to assist the client. The start-up phase took a relatively short time, one week, since it was simply marked as a formality. Everything needed to commence the project had already been in place in this week and even the requirement analysis had been started. This brings us to the next phase: analysis.

Analysis Phase

This phase started in the second week and ended in the fourth week (see fig. 5.1). The analysis process went on evenly and the customer was actively involved. It took ”only” two drafts of SRS before the customer approved it. The drafts was written by the PL with assistance of the TM, under the supervision of PM.

Design Phase

This phase lasted from the fifth to the ninth week (see fig. 5.1). In the fifth week, the project entered the design phase as soon as the SRS was reviewed and approved by the client. It was not clearly indicated who was doing what in the documents. I assumed it was mostly done by the PL and TM. In week 8 there was an email indicating a request for another TM which was, I assumed, to assist the team in the growing volume of work.

Construction Phase

This phase lasted from the ninth to the fourteenth week (see fig. 5.2). In the ninth week, the construction phase was started right after the design document was approved by the client. That particular week happened to be the busiest week, in which the highest number of hours of efforts – mostly on product related activities – was recorded.

Testing Phase

The testing phase started in the 14th week (see fig. 5.2). It was not indicated in the documentation as to who conducted the testing; however, it should be the responsibility of the team member and/or the Project Quality In charge, which was then reviewed by project leader. According to

the manual, the test should be conducted in this manner: the programmer ran the test sets and then fixed the bugs detected. For each test, he/she should determine if the unit passed or failed based on the required result specification. The observations should be recorded clearly in the test report, including other defects found using other means such as stress test, regression test, etc. Finally he/she should submit the unit test report and the item tested to the originator.

Unfortunately, week 14 was the last week recorded in the weekly project status reports, no more information was available after this period. The missing information could be about what happened during the one month free of charge support until the project sign off where all resources were de-allocated and the project was officially closed.

Delivery and Acceptance Phase

According to the project documentation, the application was delivered in the second week of December. The delivery included the complete source code and database in zipped files as required by the client and sent via mail. The delivery package was prepared by the PL and verified by the PM. According to the metrics for schedule slippage, the project was wound up by mid January 2004. Figure 5.3 illustrates the weekly effort for the project.

5.3 Quality in VMS-2 Project

This section presents the VMS-2 project in greater details with the focus on quality stages as defined the framework. Unlike the project description in the previous sections, which illustrates the chronology of the project, this section consists of vignettes illustrating the quality stages which is built on the meetings notes, chat sessions script, and other documentation.

5.3.1 Quality Practices

So far there were no quality activities prescribed specifically by the methodology. All the practices mentioned were derived from quality manual documents, and we can infer that the VMS-2 project team worked based on the guidelines written in the manuals. These quality manuals were actually the QMS of the organization which was based on the ISO 9001 standards and the SDLC/Waterfall methodology (See 3.1). This suggested that

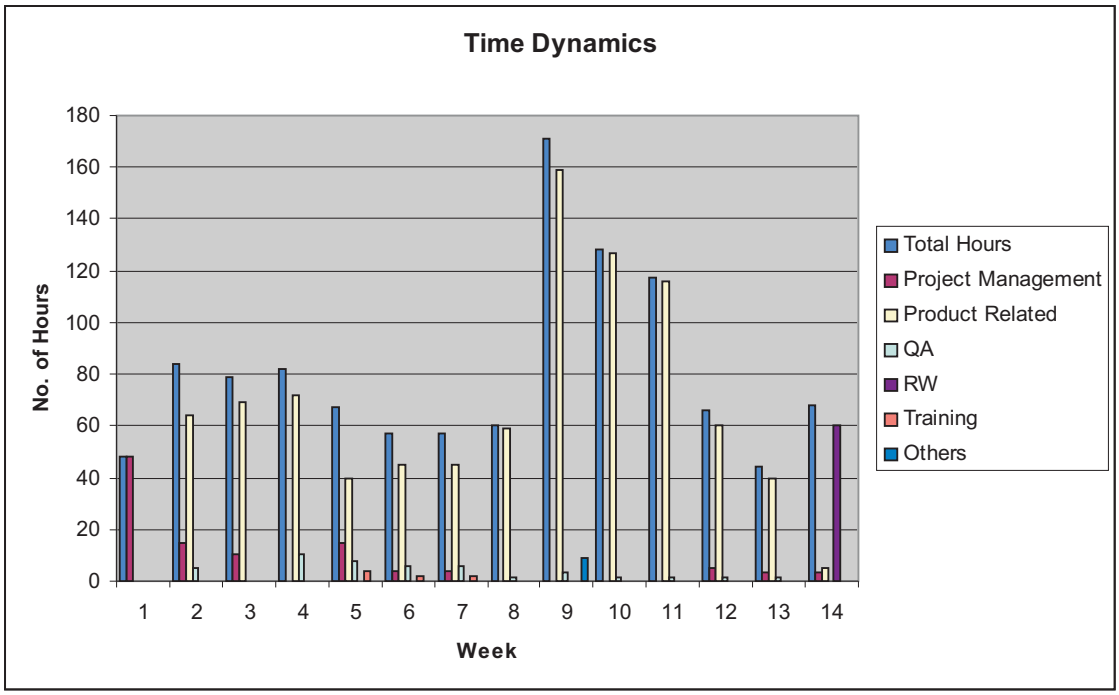

Figure 5.3: Effort spent for VMS project

this project was conducted without the need for a specialized methodology. Furthermore, it also showed that methodology and QMS could be combined as a set of standard practices for the specific project as well as the overall activities in the company.

Now, the invariant quality elements in the project, i.e. methodology (see 2.3.2) are examined as follows. About *people*, there was no information recorded about the personality traits of the people and their responsibilities. About *skills*, this was identified when analysing initial requirements and selecting team members. Training, briefing and self study were available to enhance the shortage of certain skills. In this project the training covered some enrichment in ASP technology. Only one *team* was identified for VMS-2, and they did most of the developmental activities. Some activities were conducted by external persons like for testing and auditing. No specific *roles* in the development were assigned. All team members had to be able to do any developmental task such as analysis, design, coding and testing.

About *tools*, all software and hardware needed were identified early in the startup phase. About *techniques*, the general techniques were all prescribed in the guidelines as procedures, i.e. defect handling, etc. Some

specific techniques for the project, like programming, were standardized in the team. Processes and its pre-conditions and post-conditions in this project were all following the guidelines. *Activities* – how people spend their days (Cockburn 2001) – were specified in the detailed plan and made visible to the customer with the weekly status reports. *Milestones* were specified in the project plan along with the schedule. Change of schedule, like the one happening in week twelve, was well-justified, i.e. needing more time for preparing realistic data for testing. It was also with the client's approval.

Work products were defined in the project plan as deliverables. Client deliverables included fully functional application code for the project, all source code except code for any third party or IndiSoft components, and application documentations (i.e. description of logical modules, components & controls used in the system and their integration, code of document describing the functionality of the components, installation document and test report). Internal deliverables included: Test Reports, SRS, SDD, User Manual, Final Inspection, Code review Reports, Project Delivery Note and Project Completion Report as a part of Internal Delivery. About standards, only coding standards were specified. *Quality Measures* were specified by the manuals, i.e. metrics, etc.

About *team values*, IndiSoft had its own company values which, as I observed, was available as a "quality statement", which was also written on a piece of paper hanging in all cubicles. Furthermore, I argue that the characteristics and philosophy of SDLC helped forming the team values. Such values were the importance of control, clarity and 'hard' thinking which assumed that there could be a "best solution" to solve the problem which could be engineered by following a step-by-step methodology. Now, that we saw the end result, we could safely say that they did manage to do that effectively.

The quality stages followed in the project can be illustrated in figure 5.4. The quality practices elaborated above are illustrated as the elements of the two columns on the sides, while the quality stages, which are elaborated in the next section, are illustrated sequentially confirming the use of waterfall model.

5.3.2 Quality Stages

A. Quality Definition

In this quality definition stage, we looked for various events throughout the analysis and design which denote the shaping of mutual quality per-

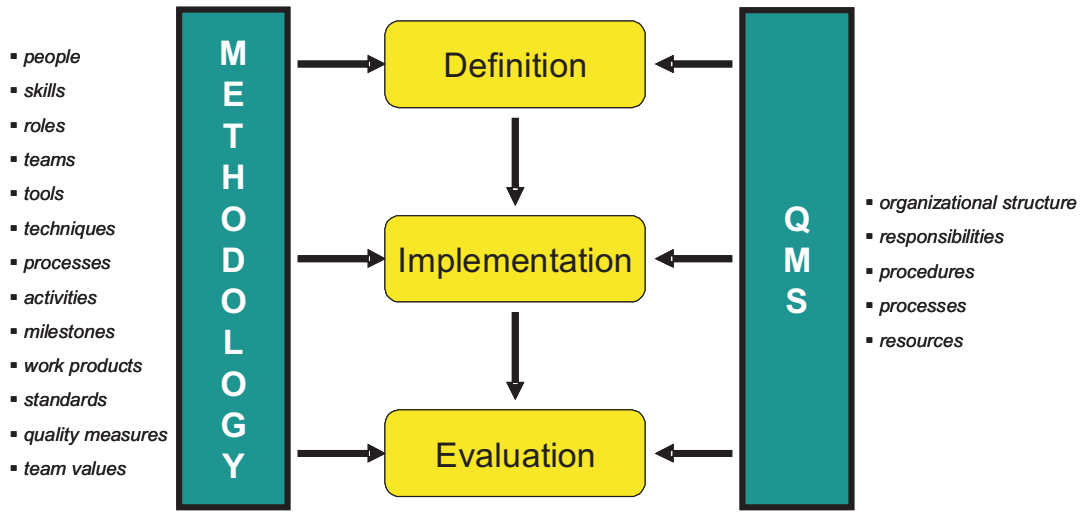

Figure 5.4: Quality Stages in VMS-2 Project

spective of both the client and the project team. In this description, the term "quality" could be exchanged with other terms like "product", etc., because here, the need for quality product is assumed to be taken as a matter of course. IndiSoft, presumably as other vendors, would surely try their best to produce a quality product, therefore all decision and actions taken here can be regarded as their effort towards quality realization. This argument also applied to the customer because we assumed that they certainly desired a quality product. Note that although both parties wanted the same thing, it did not necessarily mean that they, at this point, had the same perception of what the product should be like. This was the whole idea with the quality definition: the quality goal is shaped together by the client and the vendor.

The quality definition started as early as the contractual stage where the scope, initial requirements, responsibilities, capabilities and resources from both the client and the vendor, and review and reporting mechanisms were negotiated and agreed. As informed by a project manager, VMS-2 project was classified into SmartView category which indicated that IndiSoft had made a decision about the estimation of the effort and time required to deliver quality.

I argue that VMS-2 project was assigned to IndiSoft for several reasons. The first version of VMS was built by IndiSoft so it was a reasonable decision to work with the same vendor again so that they did not need to undergo the "get-to-know-each-other-better" period. Or it could also be interpreted that HealthCo was satisfied with the quality of the previous

work and therefore willing to continue expanding the product with IndiSoft. Either way it was a good start for the quality definition process. If we refer to the saying that "quality is when the customer comes back, not the product" [Frühauf, 1994], then this was the case for acceptable quality. The previous work was satisfactory, and the client wanted more of it. However, as mentioned earlier, we can also argue that the client came back because it was wise to give the work to someone who was already familiar with it, rather than starting all over again with a new vendor.

The *contractual stage* took a considerable time because it was the time when both parties should define the business side of the project. The customer was forced to have a refined understanding of the requirement as well as the capability and resources required. Both IndiSoft and the client were already familiar with the system because they had had experience in developing the first version. At this point, IndiSoft had already had quite an idea of how the product should behave, referring to the past development experience. And the customer, too, had the experience in using the product. We can also say that both parties were alsofamiliar with each other in terms of working styles, etc. This stage signified the initial conception of quality.

The *Project Initiation* came after the contractual stage. Here the CEO assigned a project manager (PM), introduced him to the client, and delegated the obligation of maintaining customer satisfaction and comfort. "Comfort" could be the key word for quality here, because when customer was happy, they were likely to cooperate better in the following processes, have positive attitudes towards the process and eventually accept the product satisfactorily – without complaints.

The *team building* happened in the start-up phase. Here, as prescribed by the QMS, the project leader would identify human resource requirements, reviewed by the project manager and approved by the CEO. This showed how carefully the decisions were taken. All team members had to be familiar with all general and specific aspects of the project which were given through briefing, self study materials and training. This obviously affected the quality of the work and, in turn, the product quality. In my point of view, this was an indirect way to address the people dimension in software, as was by ensuring the individuals involved had the necessary skills specific to the problems and the generally required knowledge to be able to work together as a team. Moreover, this decision was not drawn from methodological steps, but from the QMS in the organization level.

Beside the issue of selecting the right individuals, the number of people needed for the project was also a factor in quality definition because it concerned the estimation of effort required for the project. The effort plan

was derived from this agreed estimation and would be evaluated against the actual effort utilized in the project.

In one early meeting that took place in the second week and attended by the client and the team, several points were finalized, such as: client would provide examples to help the team understand the necessity of the product and its requirement; both parties agreed that new innovative ideas should be incorporated to make the best product, and the scope should not be restricted. Furthermore, technology, at this point, should not be a concern. The focus should be on the maintainability of information and ease of implementation. They agreed to add useful features to the website like site map etc. Some other technical issues were also addressed in the meeting. (Source: Summary Meeting Week 2)

Several points regarding the "rules of the game" were also established here, including: at the time of SRS approval, any issues involving major design changes would not be entertained; any questions raised to the client should be maintained as a query register that would get updated as and when the questions were raised or solutions provided; and, the possibility of maintaining vacation information was discussed.

The selection and use of technology solution in this project, such as Windows 2000, MySQL, SQL Server, ASP .NET, etc, helped shape the quality perception by imposing external constraints such as licensing requirement, inherent capability of the technology, etc; and internal constraints such as the human competence of those specific technologies that the organization had. In fact, according to the company presenation, IndiSoft had an affiliation with the vendor(s) of the mentioned technologies. Furthermore, the selection of peripheral technologies which affected the way people worked, such as version control system, etc, added to the issue. This was shown by the training request documentation which mentioned the required training on the following technology specific items, including: differences between ASP and ASP.NET; advantages ASP.NET had over ASP; programming approach using ASP.NET and some online examples.

Also, several other activities were prescribed by the QMS such as the following. Project management used deliverables, such as SRS and SDD, to capture the definition of quality. Each phase was clearly defined in the manuals with its entry and exit criteria implying that one should happen after the previous one was completed. However, in real life some activities from different phases could happen simultaneously, for example in the VMS project, the preparation of the project plan was still in progress while requirement analysis was started. This was allowed by the job distribution in the team. The project planning was the responsibility of the PL, while

requirement analysis could be started by anybody else in the team, most likely to be the team member.

Stated in the review report, the project plan of the VMS project was reviewed using a standard checklist consisting 63 item to ensure its completeness, and concluded to be satisfactory. It addressed all the necessary points about processes and procedures. Project planning activity signified the process dimension of software because it created the process roadmap that the team would follow to achieve the project goals. It also addressed the product dimension of software by mentioning and defining all components necessary to build the end product, such as tools, techniques, methodologies, etc. People dimension was also aimed well in the plan by specifying human resource aspects including back up persons and leave plans. The project plan was verified and validated through the mechanism of review and approval.

Tasks were assigned and worked on efficiently, when one task was submitted for approval, the next one was started. The first draft of SRS took only one week which was commented by the client and reworked immediately. Minor design activities, such as the layout, was also started while the analysis was conducted. It took two drafts before the SRS was approved. It was a detailed 93-page of document completed with visual aids – screen shots and diagrams – describing the complete requirements. It was designed so that the non technical customer may grasp as much as possible. The preparation of the SRS from the beginning until the client approval took place for four weeks. There were no serious problems in communicating the requirements to the client or other technical difficulties. This was another step towards the definition of quality. The client and the vendor managed to agree on the quality criteria for the end product. Layout design, previously mentioned, seemed to be treated as a side job which could be done in parallel with the other larger activities.

While the analysis activities could be seen as ”formulating the problem”, the design activities could be seen as ”formulating the solution”, including the need for tools, techniques, etc to implement it. The team sent a part of SDD for approval as soon as it was completed. By doing so, they showed that they were working on it and they gave a reasonable time for the client to study and approve it. Testing was also planned and designed at this point. Related to the subsequent programming activities, coding standard definition was compiled, code walk through was defined, pseudo code for complicated functionality was also created, etc.

According to the test plan document, the testing activity was to be conducted as followings. The plan prescribed the overall strategy to be adopted, the activities to be completed, the general resources required and

the methods and processes to be used. It also detailed the activities, dependencies and effort required. Three types of testing should be conducted, including: *unit testing* to the smallest possible unit of the application; *integrated testing* to the application after all the modules were integrated; and *system testing* in which required conditions were tested to the complete system.

The team managed to complete SDD in four weeks and shortly after the approval continue with the construction phase. In the construction phase, it seemed like VMS-2 project did not involve the client as much as in the design process, except for some clarification on several complicated modules. Probably the client was not very technical. In this case, it was safe to say that the quality definition stopped at the analysis phase. It was now up to the team, to the greater extent, to create a quality design to comply with and implement the agreed requirements. An observation for this quality stage: planning everything ahead seemed to be the maxim of quality where control was the most predominant mode.

B. Quality Implementation

This stage started at the construction phase which, since all aspects of the quality had been defined and agreed in the previous phases, "only" translated all the design into the code. The process of writing code was guided by the design document and coding standards, highly influenced by the skill of the developers which had been addressed by careful selection and training, monitored with quality assurance activities such as reviews, code walkthroughs and later on with testings.

The quality manuals prescribed software unit testing. It combined testing and debugging. So, a tester had to directly fix the bugs detected by the testing. All observations were recorded for review. The tested items and the report were submitted to the originator. It suggested that the tester was not the person writing the code, most likely to be somebody outside the team. Combining testing and debugging here was probably for practical reasons. By doing so, it would save time and resources. It was possible because all the cases, data and required results of the tests had been specified in advance, even before the construction began. The testing was conducted after the deployment of the application to the client for review.

In this quality implementation stage, several activities could be noted as being prescribed by the QMS such as the following. Project Management, as part of the QMS, prescribed the use of deliverables to monitor the implementation of quality, such as reports on Testing, Code Review, Final Inspection, Project Delivery and Completion, etc. Those deliverables

served other purposes including maintaining the visibility of the process to, primarily, the client.

There was not much to observe in this stage, presumably because the process went smoothly. The weekly status reports were maintained to make the activities visible to the client. Several emails inquiring about some detailed technical clarification were available, none indicating any serious problems.

C. Quality Evaluation

According to the portfolio in the company presentation, VMS-2 was regarded as a succesful project for the following reasons. The product worked well with the designed solution of message passing and broadcasting through a centralized web interface. It delivered the desired quality such as the increase of productivity and efficiency through rapid processing time, compatibility with the client infrastructure, usability and support for long-term investment by complying with standards.

The project was evaluated after it had been delivered to the customer, or possibly after the signing off stage. VMS project had achieved a quality level as indicated by the available metrics. It had only 11.59 % schedule slippage in average, mostly in the design stage (31.82%) followed by startup & analysis stage (25%); while the rest of the process had less than 1% schedule slippage. Those stages with high schedule slippage (design and analysis) were very crucial in defining quality. Postponing the construction until the requirements was thoroughly analyzed and the solution well-designed, seemed to be the recipe to ensure quality. Quality practices such as reviews were reported to have at an average 100% effectiveness. It means all defects reported were fixed immediately and no longer detected in the subsequent review. Rework in the project was 5.68% in average, coding took 8.86% rework and project plan 2.5%. A project manager mentioned that 5% rework was the acceptable limit. ”More than that will give us a hiccup (a common term I observed to be used across the organization, referring to troubles in project management)”, he added.

Effort variance metrics showed that on average the plans differed from the actual execution for 26.45%, with package and release stage scoring the highest 150%, followed by integration & testing 25%. Construction scored only 1.96%, compared to design (5.08%) and startup & analysis (3.13%). The project spent 16 hours of training; 6 hours were for formal training in ASP. NET and the other 10 hours were for self study on the same topics. There was also another metric called Compliance-Index, which I am not sure how to interpret.

Chapter 6

Case 2: Point of Sale (POS)

This chapter presents the second case: Point Of Sale. It starts with the project description in section 6.1, followed by the chronology showing the week-by-week progress of the project in section 6.2, which is primarily extracted from the weekly status reports. Problems concerning the data are also mentioned. Finally, the quality framework is applied here to analyse the case in section 6.3.

6.1 Project Description

The Point of Sales (POS)[1] system was a total solution developed especially for Retail Centers for a Canadian-based client, RetailSys[2]. It should provide the store retailers with an integrated accounting and management

[1]This is a common name for the software application, therefore no pseudonym used here. The term is often used in connection with hardware and software for checkouts, and in the case of variable locations, with wireless systems. POS systems evolved from the mechanical cash registers of the first half of the 20th century. Examples of this type of register were the NCR registers operated by a crank and the Burroughs registers which were opperated by a lever. These registers recorded data on journal tapes or paper tape and required an extra step to transcribe the information into the retailer's accounting system. The next step in evolution was to move to operation by electricity. An example of this type of register was the NCR Class five cash register. In 1973 new registers that were driven by computers were introduced. Such as the IBM 3653 Store System and the NCR 2150. Other computer based manufacurers were Regitel, TRW, and Datachecker. 1973 also brought about the introduction of the UPC/EAN barcode readers on the POS systems. In 1986 the POS systems became based on PC technology with the introduction of the IBM 4683. By 2005 the retail POS systems are among the most sophisticated and powerful computer networks in commercial use. Source: WIKIPEDIA (`http://en.wikipedia.org/wiki/Point_of_sale`)

[2]A pseudo name

control system. These store centers needed to be more focused on basic *Customer Relationship Managemet (CRM)*, *Inventory Tracking*, *Sales Order Processing*, *Purchase Order Processing* and *Accounting management*. In order to handle these areas effectively the system required specialized capabilities and features.

The scope of the project was to re-develop the existing application with enhanced features like implementing basic CRM, developing accounting system, online credit card validations & Electronic Data Capture (EDC). Additionally, the project team was to provide an integrated environment with bar code readers, palm device application and poll display devices, and furthermore, to make bug free application and provide customized access to the end user by providing various security levels.

As stated in the company presentation, the team worked on the solutions by addressing requirements efficiently which yielded in improved customer service, better inventory control and reduced time in order processing and accounting functions. At the same time the system was able to take care about the network down time because of the well-designed architecture. The accounting specific requirements of a POS Retailer were well handled. The team also developed an enterprise wide solution for POS terminals with connectivity to a central database. The front-end application showed the functionalities for a clerk while the store database showed interfaces for managerial functions. At the central level the data were always synchronized with different stores; and user interfaces that manipulated the data into reports were also provided.

According to the project description, POS was a desktop application developed using the DOT NET technologies completed with MSDE at the local database and MS SQL server 2000 at the central database. The environment included MS Windows 2000 Server as the operating system, and used Visual Studio.NET 2003 framework and XML parser.

IndiSoft concluded, in their project portfolio, that the client did not file any complaints. The application was live and running, satisfying the customer needs. IndiSoft were, therefore, proud of this product and strived to remain responsive to their customers. Furthermore, quantitative data were also available such as metrics, etc.

6.1.1 Project Management

The People

As mentioned in the project plan, the project team consisted of seven members: a PM, a PL, several TMs (the numbers varied throughout the project),

and another member who was responsible for business related issues and customer correspondence. This was not the composition at the kickoff. There was a takeover for PM role, and changing of team members along the way. One of the risk factors encountered, which had high severity, included team members leaving the organization or being on leave due to unforeseen reasons. To mitigate such risk, some back up persons had been prepared.

RetailSys was represented by three people: an on-site project manager, a director, and a technical person. The on-site project manager usually received all the questions and doubts from the project team in India and consulted them to the customer.

The Technology

The methodology or process model used in the project, as stated in the Project Plan, was Waterfall Model, which actually was its own version of the standard SDLC: the DLC (see 4.4). The development was conducted on Windows 2000 Professional platform using MS SQL Server /MSDE database, MSXML 3.0 Parser, Visual Studio .NET 2003, and other tools such as Visual Source Safe 6.0. It was performed by using four machines of P-III, 256-512MB RAM.

The Process

The process dimension of the project is presented as the project chronology in the next section.

6.2 Project Chronology

This section presents the case in a narrative which is resurrected from purely project documentation. The project lasted for around 30 weeks. There was inconsistent information on the project phases between the one stated in the plan and in the weekly project status. I concluded that the project has finished – underwent all the phases – like in the plan. However the chronology presented below followed the weekly status, thus, it did not have the entire phases.

Inconsistency between the weekly project status and the plan is shown in the figure 6.1 below.

<table>
<tr><th>Week</th><th colspan="3">According to plan</th><th>According to Weekly status reports</th><th>According to new schedule</th></tr>
<tr><td>1</td><td colspan="3">Startup/Initiation</td><td>Startup/Initiation</td><td rowspan="10"></td></tr>
<tr><td>2</td><td colspan="3" rowspan="5">Analysis</td><td rowspan="9">Analysis</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td></tr>
<tr><td>7</td><td colspan="3" rowspan="4">Design</td></tr>
<tr><td>8</td></tr>
<tr><td>9</td></tr>
<tr><td>10</td></tr>
<tr><td>11</td><td colspan="3" rowspan="8">Construction Stage 1</td></tr>
<tr><td>12</td><td rowspan="19">Stage 2: Construction until Deliverables</td><td>SRS sign off</td></tr>
<tr><td>13</td><td rowspan="4"></td></tr>
<tr><td>14</td></tr>
<tr><td>15</td></tr>
<tr><td>16</td><td rowspan="8">Stage 1: Construction until Deliverable</td></tr>
<tr><td>17</td></tr>
<tr><td>18</td></tr>
<tr><td>19</td></tr>
<tr><td>20</td><td colspan="3" rowspan="2">Integration & Testing</td></tr>
<tr><td>21</td></tr>
<tr><td>22</td><td>Packaging and release</td><td>Acceptance testing</td><td rowspan="7">Construction Stage 2</td><td rowspan="3"></td></tr>
<tr><td>23</td><td></td><td rowspan="2"></td></tr>
<tr><td>24</td><td></td></tr>
<tr><td>25</td><td></td><td></td><td></td></tr>
<tr><td>26</td><td></td><td></td><td></td></tr>
<tr><td>27</td><td></td><td></td><td></td><td rowspan="3"></td></tr>
<tr><td>28</td><td></td><td></td><td></td></tr>
<tr><td>29</td><td colspan="3" rowspan="2">Integration & Testing stage 2</td></tr>
<tr><td>30</td></tr>
<tr><td>31</td><td colspan="3"></td><td rowspan="6">Unknown</td></tr>
<tr><td>32</td><td colspan="3"></td></tr>
<tr><td>33</td><td colspan="3"></td></tr>
<tr><td>34</td><td colspan="3"></td></tr>
<tr><td>35</td><td colspan="3"></td></tr>
<tr><td>36</td><td colspan="3"></td></tr>
</table>

Figure 6.1: Inconsistency in POS project

Being aware of the inconsistency of the data, in this section, the project chronology is presented using the weekly status reports as in fig. 6.2 – 6.7.

Week	Activities	Effort
1	▪ The team is identified and assigned for analysis and design phase. ▪ The required development environment is setup, including the software, hardware and Visual Source Safe (VSS) as the version control system. ▪ The startup analysis is conducted; project plan and detail plan are being written	Total: 209 hours total for: project management (20), QA (4), Product Related (58), review (27) and others (100)
2	▪ The following tasks are assigned: Analysis on *Masters*, on *Order Processing*, detailed study on *Master* documentation, on *Order Tracking* system ▪ Training on Dot NET technology ▪ There was a change in project manager position	Total: 211 hours for project management (22), QA (7), product related (65), training (12), review (4), and others (101)
3	▪ Aalysis on *Masters* and *Order Processing* (completed) and sent for review ▪ Use Case for *Order Processing* (completed) ▪ *Inventory & Purchase* module analysis (started) ▪ *Sales Order Processing* is sent for review ▪ SRS & PFD for *Order Processing* (completed) and sent for review ▪ Detailed study on *Inventory flow* and *non functional requirement* (completed)	Total: 224 hours were spent for project management (40), QA (1), product related (73), training (14), review (10), and others (86)
4	▪ Analysis on *Inventory* module and *reports* (completed) and then sent for review ▪ Review of *Masters* SRS (completed) ▪ Detailed study on *Purchase flow* (completed) ▪ *Purchase* module analysis (started) ▪ Project plan is approved ▪ Use case on *Order Processing* (completed) ▪ Two TMs received training on Dot NET	Total: 174 hours were spent this week for project management (29), QA (4), product related (37), training (12), review (30), and others (62)
5	▪ Analysis & review on *Inventory & Purchase* (completed) ▪ Analysis on *Report and List review* (partially completed) ▪ Analysis on *Non Functional* requirements (ongoing – waiting for some pending answers from the client ▪ Another review on *Masters* SRS will be completed next week ▪ Analysis on the handheld device application (ongoing) ▪ Analysis on *Account* module (completed) and sent for review	Total: 161 hours for project management (21), QA (12), product related (27), training (52), review (32), and others (17)

Figure 6.2: POS Chronology

Start-up Phase

The project started in the last week of July 2003. As mentioned in the status reports, the team was created with one of them as a domain expert, in this

6	▪ Review on *Reports* and *Lists* SRS (completed) with clarifications ▪ Review on *Masters* SRS (completed) with clarifications ▪ Analysis on handheld device application (completed) with clarifications ▪ Review on analysis of *Accounts* module (completed) with clarifications ▪ Analysis on non functional requirements (ongoing)	Total: 213 hours for: PM (23), QA (21), Product related (10), training (12), SRS review (48), and others (99)
7	▪ POS process overview diagram (start) ▪ Review comments by functional expert (ongoing) ▪ Undergoing Internal audit & PMR ▪ Structuring of SRS first draft on logical sequence and functional flow (ongoing) ▪ Preparation of SRS first draft for client approval ▪ Analysis part of Payment gateway integration (ongoing) – with client' clarifications ▪ Analysis of some other non functional requirements, e.g. preparation of a general template which will have common specs for the application (ongoing) ▪ Analysis for developing a handheld device application (ongoing) ▪ VB.Net training program (completed) ▪ Help files development (ongoing)	Total: 136 hours for PM (25), QA (29), product related (10), SRS review (38), others – including self study (34)
8	▪ Work on the pending clarifications sent by the client on some of the third party devices like Credit card processing, understanding Payment Gateways PDF's sent by client like IC VERRIFY etc. ▪ Work on the screen shots and design of hand held application ▪ Work on the integration issues of Pole display units ▪ Identify the common components with other development teams to enable reuse ▪ Create templates for some of the sample screens ▪ Develop user interface standards document ▪ Work on help files design template	Total: 137 hours for PM (20), QA (5), product related (36), and others (76)
9	▪ Work on POS user interface standards and send to the client for approval ▪ Identify POS components and send to the management for review and approval ▪ Work on requirement specification for handheld device application and sent to the client ▪ Work on the clarifications received from the client	Total: 139 hours for PM (18), QA (3), product related (53), training (1), SRS review (4), and others (60)

Figure 6.3: POS Chronology (ctd.)

10	▪ Work on design template for the handheld device application ▪ Work on the clarifications received from the client ▪ Work on credit card integration issues and barcode reader	Total: 124 hours – PM (21), QA (4), product related (29), training (3), SRS review (6), and others (61)
11	▪ Final version of SRS is sent to the client and wait for client approval ▪ Use Case diagrams for *Order Management, Purchase, Inventory* and *Accounting* modules (completed) ▪ Handheld device application prototype (ongoing) ▪ Conduct brainstorming sessions on *Application* solution architecture (ongoing)	Total: 117 hours for PM (35), QA (8), product related (17), SRS review (2), and others (55)
12	▪ Two of the customer representatives visit IndiSoft ▪ SRS is approved ▪ Client reviews and approves design template on masters and user interface standards documents ▪ Client is to send the required 3rd party devices – agreed during the visit. ▪ Schedules are updated ▪ Team sends the POS presentation (includes phase wise schedules) and Minutes ▪ Finalize POS solution architecture ▪ Work on Context Analysis diagrams and DFD's for POS	Total: 120 – PM (30), QA (10), product related (20), training (3), and others (57)
13	▪ Initiate the design phase as the SRS was signed-off on the previous day ▪ Context Flow diagrams analysis and DFD's (ongoing) ▪ Entity Relationship diagrams part of the design template (ongoing) ▪ Components design part of SDD (ongoing) ▪ Work on certain screen images, screen objects, actions and reports (a part of Human Interface Design of SDD document) ▪ Work on the architectural design of POS application	Total: 143 hours – PM (13), QA (18), product related (57), training (1), design review (1), and others (54)
14	▪ <same as the previous week>	Total: 102 hours – PM (22), QA (13), Prod related (19), review (1), and others (47)
15	▪ <same as the previous week>	Total: 128 hours – PM (23), QA (19), prod related (61), review (1), and others (26)

Figure 6.4: POS Chronology (ctd.)

16	▪ <same as the previous week>	Total: 91 hours – PM (14), QA (6), product related (6), and others (62)
17	▪ Context Flow diagrams analysis and DFD's (completed) ▪ Entity Relationship diagrams part of the design template (ongoing) ▪ Logical Database design for *Core* POS, *Inventory* and *Purchase* (ongoing) ▪ Finishing some screen images, screen objects, actions & reports (a part of Human Interface design of SDD document) ▪ Component design, architecture design, and logical database design (completed) ▪ Deliver the SDD to the client	Total: 119 hours – PM (12), QA (4), product related (5), and others (99)
18	▪ Review the logical and physical database design to have better optimization (ongoing) ▪ Completing the screen images, screen objects, actions and reports (a part of Human Interface design) ▪ Work on some more screen designs to have good control during development stage ▪ Complete the executable creation for some the important screens with two options to the client and sent it for the clients approval	Total: 78 hours – PM (11), QA (5), product related (19), and others (43)
19	▪ <Same as the previous week> ▪ Send SDD to client	Total: 86 hours – PM (18), QA (2), product related (19), and others (48)
20	▪ <same as the previous week>	Total: 73 hours – PM (11), QA (2), product related (1), and others (60)
21	▪ <same as the previous week> ▪ Complete the identification of the XML format that was to be adopted for the transfer of data ▪ Complete a few of the components to make development easier ▪ Design project template according to the solution architecture	Total: 55 hours this week: PM (9), QA (2), product related (1), and others (46)

Figure 6.5: POS Chronology (ctd.)

22	▪ <same as the previous week> ▪ Face internal audit	Total: 65 hours: PM (11), QA (2), product related (1), and others (56)
23	▪ <same as the previous week>	Total: 63 hours for PM (9), QA (2), product related (1), and others (52)
24	▪ <same as the previous week>	Total: 64 hours: PM (9), QA (2), product related (1), and others (64)
25	▪ Review the logical and physical database design to have better optimization (ongoing) ▪ Completing the screen images, screen objects, actions and reports (a part of Human Interface design) ▪ Work on some more screen designs to have good control during development stage ▪ Complete the executable creation for some the important screens with two options to the client and sent it for the clients approval ▪ Complete the identification of the XML format that was to be adopted for the transfer of data ▪ Complete a few of the components to make development easier ▪ Design project template according to the solution architecture	Total: 115 hours: PM (9), QA (10), product related (26), and others (71)
26	▪ <same as the previous week>	Total: 84 hours: PM (9), QA (2), product related (3), and others (66)
27	▪ <same as the previous week>	Total: 69 hours: PM (9), QA (2), Product related (24), and others (35)
28	▪ <same as the previous week>	Total: 72 hours: PM (9), QA (10), product related (26), and others (57)

Figure 6.6: POS Chronology (ctd.)

case: Accounting. At this point, the team had done the pre-analysis on the

29	▪ <same as the previous week>	Total: 101 hours: QA (16), product related (80), and others (31)
30	▪ <same as the previous week>	Total: 86 hours: QA (10), product related (54), design review (26), and others (6)

Figure 6.7: POS Chronology (ctd.)

requirement specification from the customer. The Project Initiation form was not available, so I could not inform the formal details of the projects, the expected start and end dates, the estimated effort in person-month and the resource requirement. That information was scattered elsewhere in the documents, i.e. project proposal which was dated before the initiation. According to the history of the document, its preparation started two months before. It indicates that the interaction with the client had started much earlier.

Right after the project was initiated, communication with the customer took place frequently. Chat sessions happened almost everyday. Questions and answers also took place via emails. The questions were technical and detailed. Three customer representatives were involved in the communication; sometimes separately and some other times together.

Analysis Phase

According to the weekly status reports, this phase started in the second week, and ended in the eleventh week (see fig. 6.2 – 6.5). Unlike the status report in VMS project, the one used in this project also mentioned the identified issues/concerns, and risk. For example, in one of the weeks in this phase, the issue was that the client had not sent the information on the Canadian accounting system. The team identified risks such as: (1) bar code reader and payment gateway integration were still in gray areas; impacting schedule; critical by 30 August; (2) availability of bar code reader to the team; affecting schedule; to be provided by client; and (3) deployment environment; affecting schedule; to be provided by client.

The weekly status reports also mentioned customer input which worked as an acknowledgement (that the input had been received) as well as a re-

minder for the customer of its promises and obligations. For example, in the seventh week, these customer inputs were recorded:

> "some answers were pending from the client like *non functional* requirements, multiple currency support, pole display, etc; some information on bar code reader and payment gateway integration had been provided but the team still needed to get full information on this; availability of bar code reader to development team – any emulator software would be recommended in place of a physical device; availability of handheld device specific information like (credit card related IC VERIFY software) to cross check whether the suggested environment was compatible; and handheld device availability for the team to deploy and test the hand held device application." [Weekly status report, week 7]

This phase ended when the final version of SRS was approved by the client. It was the compilation of all the modular analysis into one single document.

Design Phase

This phase started in week 12 throughout the week 30. According to the metrics data, the construction phase was delayed for more than a month. The cause of the delay was the late approval of SDD by the client. However, it was interesting to notice that while waiting, the weekly status reports kept reporting activities spent for this project, which sometimes merely a repetition of the previous week with different number of hours.

Figure 6.8 displayed the weekly effort for the project. The data was taken from the internal status report – it is the only available source despite its inconsistency mentioned previously – and PMR documentation. Following the procedures, the team grouped their activities into: project management (PM), product related (Pr), Quality Assurance (QA), Review Meetings (RW), training (Tr) and others (O).

The first 6 weeks, the weekly total effort exceeded 150 hours; most of them even passed 200 hours. I believed this was because requirement specification was the hardest and most intensive part of the project. In each week throughout the project, most effort was spent on product related and other activities. It was unclear why activities categorized as "others" was predominant, implying that the team was doing activities which did not belong to the rest of the categories. What could that be? "Other" activities maintained above 50 hours in most of the weeks. Is

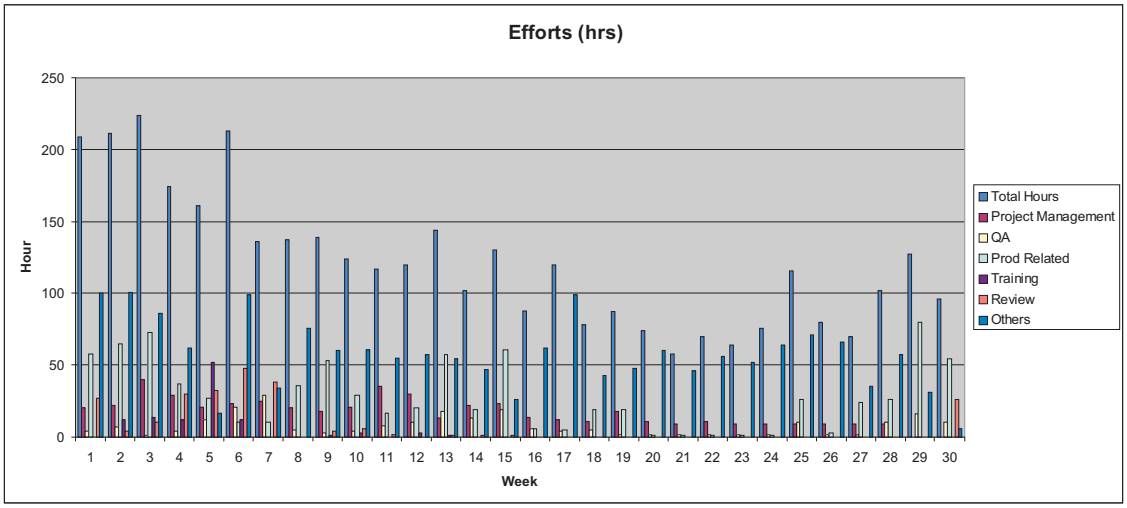

Figure 6.8: Effort spent in POS project

this the source of overhead cost? These questions could have been addressed if the research had been conducted in a different way. The activities categorized as Project Management, QA and review never exceeded 50 hours/week. They mostly maintained far below that figure.

6.3 Quality in POS project

This section presents the POS project in a different manner by using the quality framework (see 2.5). Unlike the project description which illustrates the chronology of the project, this section consists of vignettes illustrating the quality stages which are built on the meetings notes, chat sessions script, and other documentation. Due to the inconsistency of the data, as shown in fig. 6.1, we can describe the quality definition stage in greater details than the other stages.

6.3.1 Quality Practices

We can start observing the project by going through the invariant elements of quality such as the methodology elements (see 2.3.2) as follows. Firstly, there was no information recorded about the personality traits of the *people* and their responsibilities. Required *skills* were identified when analysing initial requirements and selecting team members. Training, briefing and self study were available to enhance the shortage of certain skills. In

this project the training covered some enrichment in Dot NET technology. Only one *team* was identified for POS project, and they did most of the developmental activities. Some activities were conducted by external persons for testing and auditing. No specific *roles* in the development were assigned. All team members had to be able to do any developmental task such as analysis, design, coding and testing. This project used a domain expert at the beginning of the project to support the requirement engineering activities.

About *tools*, all software and hardware needed were identified early in the startup phase. About *techniques*, the general techniques were all prescribed in the guidelines as procedures, i.e. defect handling, etc. Some specific techniques for the project, like programming, were standardized in the team. Processes and its pre-conditions and post-conditions in this project were all following the guidelines. *Activities* – how people spend their days (Cockburn 2001) – were specified in the detailed plan and made visible to the customer with the weekly status reports. *Milestones* were specified in the project plan along with the schedule. Change of schedule or delay, which happened significantly in the project, was justified well, i.e. the team was ready to wait for the client's approval.

Work products were defined in the project plan as deliverables. Client deliverables included fully functional application code for the project, all source code except code for any third party or IndiSoft components, and application documentations (i.e. description of logical modules, components & controls used in the system and their integration, code of document describing the functionality of the components, installation document and test report). Internal deliverables included: Test Reports, SRS, SDD, User Manual, Final Inspection, Code review Reports, Project Delivery Note and Project Completion Report as a part of Internal Delivery. About standards, only coding standards were specified. *Quality Measures* were specified by the manuals, i.e. metrics, etc.

About *team values*, IndiSoft had its own company values which, as I observed, was available as a "quality statement" written in a piece of paper hangin in all cubicles as mentioned in the previous case study.

Figure 6.9 shows that this project experienced a significant delay from quality definition stage to the implementation due to waiting for approval of the SDD – the final quality definition to implement. It implied that, in this project, one quality stage could not be started before the preceding one had finished.

Judging from the *process dimension*, some observations were noted as the following. Quality project and product were interchangeable. Quality project was present only when the product was delivered as desired and

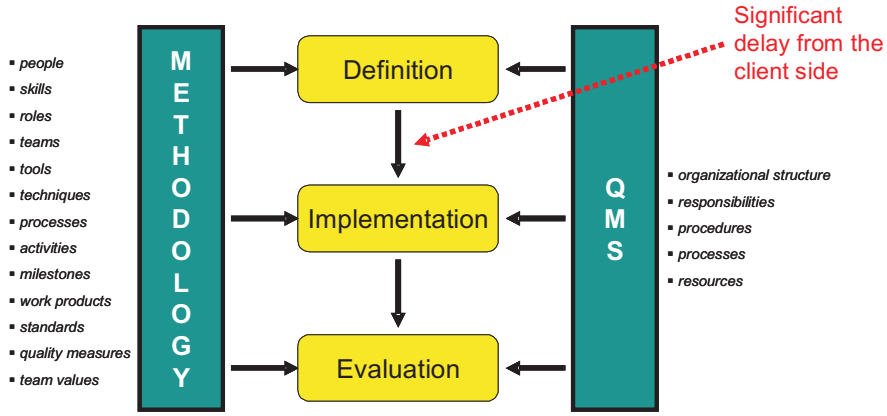

Figure 6.9: Quality Stages in POS Project

as planned. The quality of POS project depended notably on the client. It showed clearly that quality was not an objective entity, it was a matter of compromise between the client and vendor. The project was considered successful despite the long delay caused by the client. In this case the client was not entitled to complain, and the vendor could not be complained either because they had done their part well. It would have been different if it happened otherwise. So, the same condition, i.e. delay in a certain phase of the development, could to be perceived differently in terms of quality.

This case signified that quality was in the eye of the beholder. The vendor had obviously regarded this as a success. The client, however, did not necessarily share the view since, I presume, that they had to pay for the consequence of the delay. The end system might be of their desired quality, but the overall project cost them more than it should have been. It might be seen as a failure from their side since they had failed to follow the agreed plan, which cancelled them the right to complain, at least, about the time and cost of the project. The absence of complain could be seen as one of the indicators of quality. Again, quality was hard to define, but its presence or absence could be easily recognized. Quality (judgement) was about acceptable compromise.

Judging from the *product dimension*, this project achieved its quality goals. All workproducts, for internal as well external purpose, had been delivered and no complaints had been noted about it. The product oriented approach tried to guide the quality improvement by making the product quality explicit; and this was exactly what the team had done. They managed to produce documents indicating the product quality, such

as SRS and SDD, in an exhaustive manner, by putting in all the details and focusing the informative side with diagrams and examples. This had impressed the client.

To judge from the *people dimension* was relatively more vague because there was no "hard" indicator about it. Since interpretive approach is used here, I interpreted that the team had achieved the quality goal in the people dimension for the following reasons. With all the activities described above, the team members managed to collaborate and produce the result as planned, they managed to maintain the communication with the client in a close, polite and accommodative manner. Since I had no data about the people in this project, all I could do was relating this with the observation I had done in the fieldwork. I noticed that, in general, people in the company had a high work morale, discipline, positive attitude and were accommodative to others. The team also managed to reduce the human tendency to produce error in his actions by applying quality assurance in the project. This was shown in the evaluation stage by the review effectiveness of 100%.

6.3.2 Quality Stages

A. Quality Definition

The earliest contact made by the client, RetailSys, was recorded 5-6 weeks before the project was initiated. RetailSys sent an early requirement specification of POS, describing: what POS was, the technology, overview of the system, and the important modules including *Order*, *Customer*, *Inventory*, and *Tools/Reports*. At that early stage, for example, the client specified the technology as followed:

> "It should provide the benefits of Windows programming technology, true client server database design, and be 100% open to offer a business solution that is highly scalable, secure, recoverable and very fast. The system must be very easy to bring back up if it crashes and should also analyze the data and self correct after a crash. The system should allow the clerk to keep selling even when the network is down. (Early Specification Document)"

The specification, as quoted above, showed specific technologies which the client perceived as the "good" one for their cause. The selection, I argue, would also determine the outcome of the project later on because

of the limitation imposed by the technology. It also mentioned several quality characteristics to be applied to the system, such as scalable, secure, recoverable and fast.

After studying the initial requirement specification from the customer, IndiSoft followed up with a project proposal which consisted of the following items: *scope*, which was derived from the client requirements, consisting of the main features, enhanced features and additional features; *technology* as previously elaborated in the previous section (see 6.1.1); their *understanding* of the modules, proposed features, and solution for phase 1 and phase 2; and an *effort estimation* of 3126 person hours for 28 weeks. The project was then initiated, and a team was built consisting of 5-6 people (the number varied throughout the project). It included one person with the domain knowledge. These activities illustrated the early exchange of information between the client and vendor in shaping or negotiating the basis for quality achievement.

The project plan was approved early in week 4. This document established the "rules of the game" officially. Schedules was mentioned in the project plan and referred to the detail plan. Concerning communication, the client and the project team agreed with two business days response period.

> "Response period from both sides (Client and Project Team) should be two business days – (Project Plan)"

The early communication described above was just the beginning, in the weeks to come POS team and RetailSys representatives communicated intensively, mostly by chatting via public messenger application and emails. The result of each question-and-answer session was documented and made available to the whole team in the point-to-point basis that is easy to follow, like this.

"

- PM acknowledged the receipt of answers to the questions that were sent on 21 July 2003. However some of these were not answered. Confirmations from the client were pending. Client1 will get confirmations for these.
- PM asked the details of tint code. Client1 will get the details of Tint code from the client
- Client1 confirmed that tint codes (formula) would be stored in customer history. Client1 would confirm the availability of the same formula to other customers from the client.

- Client2 reminded about the Project Plan and résumé's of team members. PM informed him that they would be sent after approval from group manager. PM will take approval and send Project Plan and résumé's.

(Summary Chat in Week 1)"

As prescribed in the manuals, it was a common practice in the organization that only PM had direct contact with the customer. This was to ensure that there was no distortion in communication. This practice had been tacit to the organization, but not so explicit to the client. It was noticed quite late (in week 9) by the client representative.

"client@hotmail.com says:
Raju (not the real name), are you the only person on the chat from IndiSoft?
PM@hotmail.com says:
yes.
client@hotmail.com says:
OK just wondering
(Chat session in Week 9)"

Questions and doubts from the other team members were compiled in a document and sent to the client via email. The client would address the questions or put on hold when he/she was in doubt. All these documents could be a reference for later purpose. The questions and answers documents are like this:

"4) What are the processes involved in Order Deposit?
Will confirm with client tomorrow

5) Can there ever be a refund of deposit ???
Yes, for example a customer orders a certain wall paper and puts down a deposit. The store owner later finds that the product is discontinued so refunds the deposit.

6) What are the steps involved in Purchase Order?
Will confirm with client tomorrow

7) How do you handle enquiries ?
Don't understand, please rephrase

('Answer' document in week 1)"

The team gave a positive self assessment in their attempt to define quality. This was recognized by the client as well as the IndiSoft management. The quality of the work so far was appraised in the in Project Quarterly Review in week 11 which was attended by the CEO, Group Manager and all team members.

All the requirements – as a result of a intensive and iterative process of specification – were compiled in a single and complete SRS document. It took some weeks to produce such a document. The team and the client negotiated and agreed on how long they would spend time for it, which was another instance of process transparency and client involvement in every step of the journey.

"client@hotmail.com says:
Do you think 5 weeks is enough time for the SRS?
PM says:
for the preparation of entire SRS, 5 weeks is enough
PM says:
after having all the clarifications
client@hotmail.com says:
Good - I just have to get all of the answers to you quickly
PM says:
yes, that will be a great help to me and the team
(Chat Session in week 6)"

Yet, after the agreed time passed, the team impressed the client by producing a long and detailed SRS, as shown in the chat session below. I argue that this impression had a positive impact on the client's perception of IndiSoft.

"client2@hotmail.com says:
i think i must have used a whole tree to print out this SRS document!
PM@hotmail.com says:
It's a detailed SRS
PM@hotmail.com says:
How is the overall SRS
client2@hotmail.com says:
very well done....good detail and examples
PM@hotmail.com says:
Thank you very much
(Chat Session in Week 9)"

As mentioned earlier, in the weekly status reports, some issues and risks were identified as a reminder. The team kept mentioning the issues before it was resolved in the status report along with the critical time which might affect the project schedule. Besides, they also proactively asked about those issues during chat session as below:

"PM says:
how the integration with the Bar code Scanner should take place pl
client@hotmail.com says:
I will email you info on the bar code scanner tomorrow
PM says:
great, thanks
PM says:
we need some emulator software in place of a physical barcode reader, what's your idea
client@hotmail.com says:
That makes sense. I will have info tomorrow
PM says:
thanks
(Chat Session in week 6)"

During the project, the weekly status reporting had been a good practice to maintain transparency of process and client involvement. Even the clients showed their appreciation of it – especially the format of the report – as shown in the chat session below:

"PM says:
it's nice that you have seen our project management tool too
client@hotmail.com says:
I was going to say I really like your weekly status report format
PM says:
oh! thanks for liking it
(Chat Session in week 5)"

The PM also tried to introduce the other project management tools and practices such as chat summary, Q&A compilation, etc and encouraged the client to use them, as illustrated below:

"PM says:
fine, how r u
client@hotmail.com says:
Good, yourself? I will have questions and answers for you tomorrow.

Tonight I have nothing
PM says:
great, that's fine Mary (not the real name), we have actually kept all the queries and
PM says:
chat minutes in Project Managemet tool and sent you thru mail too, pl see them
client@hotmail.com says:
I will look. Do you mind if we sign off until tomorrow?
(Chat Session in week 5)"

During the creation of the design document (SDD), the communication was not as intensive as before. It seemed like the SRS creation (analysis) was the culmination of quality definition stage with heavy client involvement. According to the project documentation, the SDD was finally produced, and then, there was a significant delay between the design and the construction process as mentioned in section 6.2.

B. Quality Implementation

As we have mentioned earlier, the status reports for the phases after week 30 (design) were not available. Very little information was available elsewhere documenting what happened in this stage, as the following.

"

- So far coding has been completed for 28 Master Screens.
- Coding for Inventory Module will be completed by the end of the Month.
- Coding for Core-POS Module started 2 days ago
- Purchase Module need to be started with in 4 days.
- Schedule for Phase-I has to be Prepared by the PL (Summary Meeting week 31)

"

It showed that construction had been going on for quite some times even though until week 30 the weekly status was still labeled "Design Phase".

C. Quality Evaluation

According to the portfolio in the company presentation, POS was regarded as a succesful project for the following reasons. Beside the qualities mentioned in 6.1, the metrics data showed that the average schedule slippage was 0%. The construction phase was actually delayed for more than a month. This was due to delay from the client side causing the team to wait for the approval of SDD. Therefore, the management considered this as zero schedule slippage. The construction phase was actually finished much earlier than the allocated time: - 69% schedule slippage. It showed how well the team performed. It was also probably the reason why weekly status report was not available after week 30, namely the rhythm of work had changed due to the delay. Furthermore, as in VMS-2 project, when SDD had been produced, the construction could go fast and smoothly.

Review effectiveness was 100%: one review found 9 defects which were all fixed and passed the subsequent review. There was no rework noted in the metrics. Effort variance was also 0% noted. The team used 437 hours for on-the-project training, including formal (90 hours) and informal self study (347 hours).

Chapter 7

Discussion

This chapter is divided into sections which are addressing the research questions posed earlier. Section 7.1 presents the similarities and differences of the two case studies. Section 7.2 discusses the quality determinants in GSW. Section 7.3 suggests several best practices to global software work with a focus on quality achievement.

7.1 Comparison of the two case studies: VMS-2 and POS

7.1.1 Similarities

Both the VMS-2 and POS projects used waterfall methodology and followed the quality manuals in determining their *rules of the game*, i.e project management, work products, processes, customer communication etc. The teams did not tailor the methodologies and processes according to the problems, which implied that they assumed the two problems could be solved in a standard way.

The analysis showed that they had followed the same pattern in the quality stages which are closely related to the choice of methodology and process model. It also showed the blend of methodology and QMS. The waterfall methodology had been institutionalized and standardized in the QMS for all (most) projects in the organization.

Both projects used most of the technologies and platforms from the same vendors. The success of the two projects indicated their competence in those technologies.

7.1.2 Differences

The obvious differences between the two projects were the size and the complexity of the problem. The other differences were derived from them. Firstly, they were categorized in different division (practice group) in the company: VMS-2 was in SmartView, while POS was in SmartBiz (see 4.2). Unlike VMS-2, POS project required more people including a domain specialist. Secondly, communication with client was more frequent and intensive in POS than in VMS-2. Post mortem analysis of POS project was more illustrative in terms of the communication between client and vendor. Chat sessions were more frequent and well-documented. I believed it was because the size and complexity of POS project required more intensive communication. Finally, more risks and issues were identified throughout the POS project than VMS-2.

7.2 Quality determinants in GSW

This section answers the first reseach question by discussing the quality determinants in GSW context. Thus far, I still believe that quality is hard to define but its presence or absence is easier to recognize. I departed from the standpoint that quality had been achieved in the two case studies previously presented and analysed, and then tried to specify and discuss the quality determinants found in those cases. In this thesis, quality determinants refer to the important factors on which software practices in these projects are more focused in order to achieve quality. Those can be the positive factors which the software team must ensure its existence, or they can also be the negative ones which must be avoided. *These factors are derived only from the practices observed and analysed in the thesis, therefore it cannot claim a broad generalization.* Moreover, some of those factors confirm findings from previous studies (see 4.1.3).

Quality is in the eye of the beholders, which primarily here are customers. The customers are the first to judge the quality of the project they initiated. The indicators of the judgment are the customer's acceptance, appraisals and complaints. The presence of the first two and the absence of the latter imply quality achievement. Thus, the following quality determinants are derived from the practices which lead to customer satisfaction.

A. External Factors

7.2.1 Temporal Engagement with Client

In many outsourcing projects, the project teams (vendors) were engaged with the client only as long as the project development is going on, with possible time extension after the product was delivered and accepted. Beyond that was practically out of their concern. There was not much for the project team to do when the product – their brain-child – was operational in the client organization. Nevertheless, being operational was actually the ultimate test for the product's quality, namely to show its innate excellence (transcendence view of quality) and its actual fitness for purpose (user-based view) (see 2.2).

The temporal engagement with client in the GSO context, in which clients were beyond national boundaries, made the process dimension of software development more predictable, and therefore feasible, to be the basis for work arrangement to achieve quality than the other two dimensions. Product dimension was less predictable because there was practically no way to know the product quality before it was being operational. People dimension was even more difficult to foretell, as difficult as to predict social interaction between human inviduals in general. I believed it was the reason why roles in the project team in the case studies were not defined by the development activities, such as analyst, database designer, user interface designer, programmer, but by the generic project management activities, namely project manager, project leader and team member. By doing so, it was easier to allocate people to different projects.

Nevertheless, engagement between the client and the vendor was prerequsite to the project, and therefore should be established. The engagement manifested in different ways in each quality stage. In the quality definition stage the engagement was manifested in the formal contractual agreement which stated the scope of the project and the required conditions from both sides; project plan which included the developmental stages and milestones of the project; SRS which showed the agreement on what problems to solve and SDD which shows the agreement on what solutions to develop. In the quality implementation and evaluation stage, the engagement manifested in the execution of the engagement defined in the quality definition stage. Failure to meet the agreement would cause problems to the project, such as the delay of SDD approval in POS project which had caused a significant schedule slippage.

The temporality of the engagement was closely related to the cost of the project. Time was the main unit for calculating the efforts and the cost of

the project. Early in the project, an effort estimation was determined and agreed. A plan was then made based on this estimation. For the vendor, this estimation was important because it indicated their professionalism and secured them the deal with the customer. For the customer, this estimation was important because it indicated the approximation of cost they had to pay.

7.2.2 Visibility of processes

The visibility of processes referred to the possibility for a customer to monitor the activities at the production end, despite the challenges posed by the time differences and the geographical separation across the national boundaries. Being able to "see" the process, the customer would trust the vendor, be engaged in the project more closely, and eventually, together with the team, help shape the end quality.

The visibility of processes facilitated the client and the vendor in the quality stages, especially the quality definition. In the two case studies, this constituent was mainly provided by the vendor, as indicated by the provision of communication infrastructure and, more importantly, the willingness of the project management to open themselves – embodied in different practices such as documentation and reporting, as elaborated in the next section. The client might have some ideas about this factor, but they did not always have the power to impose it to the client by, for example, providing means of communication or enforcing related work practices. Referring to the software dimensions, this determinant was obviously a focus on the process dimension because from that perspective, processes were the most (if not the only) visible – i.e. available, accessible and reportable – elements throughout the project.

7.2.3 Global Communication

The global communication here referred to the ability and mechanism to exchange information throughout a project between a vendor and a customer which were globally separated. Communication was indeed an important element in software development. It occurred, not only between the team members, but also between the team and the customer. This factor should be mutual, provided that the rules of the game were agreed in advance. Loosing the richness of face-to-face communication in the global setting (different location, time zone and culture) was the most widely known challenge in GSW setting. Globally distributed teams must, there-

fore, establish a way to overcome this challenge.

In our case studies, only the customers were separated from the rest of the project teams. The project teams from the vendor site were co-located, and therefore, they were able to benefit from the use of rich face-to-face communication means. However, since the customers played an important role in judging the quality of the project, it was important for the vendor to provide the best communication mechanism as possible. This factor was closely related to the previous one, *visibility of processes*, as the communication process is essential in making the development process visible to client. Furthermore, customer communication also took us to the next important, and still closely related, factor: *customer consent*.

7.2.4 Customer Consent

The customer consent here referred to the customer's approval and agreement of what is done and proposed by the vendor. This determinant was political and partial in nature because quality determination could be seen, to some extent, as a power struggle in which two parties (vendor and customer) managed, contrived or dealt to achieve common goals while protecting their own interests.

As shown in the POS project, obtaining customer consent was mostly for the benefit of the team. The POS team could not be blamed of the significant delay which occured (see 6.2). When done iteratively in the beginning and end of every activity, the agreement bound the customer to the current result before they move to the next activity. It also avoided blame in different circumstances which might inflict the vendor's interests.

***

B. Internal Factors

Beside the external factors mentioned above, quality was also determined by the internal ones which are so apparent in the two projects, such as:

7.2.5 Control

It referred to the ability to exercise restraining or directing influence over all the elements involved in the development, such as scope determination. In both projects, the teams managed to define the overall scope in the beginning, followed by the more detailed ones in the subsequent stages. Having a clear scope in each development phase allowed them to plan

activities and risk mitigation, and eventually, influence the quality of the outcome.

7.2.6 Predictability

It referred to the ability and necessity to declare or indicate as many things as possible *in advance*. This determinant gave a way for the teams to anticipate any future issues and risks, and to alert the client in a timely manner. This determinant was closely related with the previous one: predictability was made possible by the team's ability to control or direct influence over the development elements. I argued that this factor, as much as the previous one, was principally caused by the adoption of the waterfall methodology.

7.2.7 Technical Competence

It referred to the ability to use the tools, perform the techniques and apply the methodology in any given stage of development to produce desired results. It also included the way of making the competence available in the organization through trainings, etc. The use of certain tools and technology in the project imposed their inherent constraints to the solution which, in turn, influenced the quality of the product.

7.2.8 Accomodativeness

It referred to ability and willingness of the vendor to adapt to the client's non technical needs and conditions. This factor became more important due to the conditions imposed by the global work setting, i.e. different time zones, holidays related to local culture, etc. Working hour adjustment, including willingness to work on weekends, was an example of accommodativeness which was observed as one of the stroing points of IndiSoft.

7.2.9 Synergy between Methodology and QMS

Based on the two case studies in IndiSoft, I realized that methodology tailoring process was rather unlikely to happen in this organization. A methodology consisted of process, method, tools and philosophical foundation (Avison & Fitzgerald, 2003). Both projects showed the use of the

same philosophy, namely the underlying theories and assumptions of waterfall methodology, and the quite similar sets of processes which only varied in terms of length and complexity. They varied only in the use of tools and technology which was specifically required by the problem domain. These variations could not be considered as methodology tailoring.

Instead, the methodology was standardized and combined with the external standards, such as ISO, to make an overall quality management system for the organization.

7.3 Best practices

In this section the practices which found in the case studies, especially those focused on the quality determinants, are discussed. They are the best practices which can be suggested for practical purposes.

7.3.1 Documentation

Documentation was the only tangible way of representing the software and software processes. Referring to the temporal engagement with clients (see 7.2.1), process dimension was the most, if not the only, feasible way on which to base the software work. Therefore, documentation became a very important practice. However, this was not a new thing in software engineering. The problem was actually how to create a useful documentation which served the cause of the team as well as the customers. It should not be an overhead activity for the developers, and it should be clear and good enough for the customer to understand regerdless of their level of technical knowledge. Concerning the main topic at hand, two major documents – SRS and SDD – played an important role in achieving the goal of producing quality software.

The teams compiled those documents in an exhaustive manner attempting to cover the problem and solution domain as much as possible. This way they helped the customer to be more involved in the development process. It also reduced the uncertainty caused by the inherent intangibility of software. Other documentation, such as test plans & reports, code walkthroughs, etc, facilitated the visibility of process (see 7.2.2).

7.3.2 Reporting

The reporting practice was not new either. The question was how to do it effectively to overcome the non co-located situation. In the case stud-

ies, it was shown how effective a weekly report could be. The customers appreciated it because it was brief and clear. I personally agree with that since those weekly reports made it possible to reconstruct the project long after it had finished. The report contained the following elements: what had been done, what to do next, risk and issue, and alerts. The internal version of status reports also registered the number of hours spent in the week.

7.3.3 Team building

The teams in the case studies always consisted of two leaders (PM and PL) and one or more team members (TM). The assignments were not based on development roles such as analyst, designer, developer, tester, etc. Instead, each team member, sometimes even PM and PL, should be able to function as any of those roles. It enabled the team to assign development tasks more dynamically and independent of the people.

The job distribution of the two leaders was a good practice to overcome the distributed work. The two-layer team leadership enabled more effective exercise of control both internally and externally. Communication was the key here. The PL communicated more with the team members and managed the day-to-day life of the project. This gave more space for the PM to focus on customer needs. Communication with client was handled primarily by the PM.

7.3.4 One channel communication with client

Without the richness of face-to-face communication, the interaction between the client and the vendor became more challenging. In the worst case it could fall into communication disasters – complication, distortion, and misinterpretation – which would harm the project. Problem of identity and the limitation of the medium were some of the challenges in this faceless communication. So, to avoid confusion of identity it was better for the client to communicate only with the project manager during the project. Shown in the chat script, after communicating for some time, the relationship between the client and the PM become more personal. Yet, the client realized, only after week 9, that she had been talking only with the same person for so long.

7.3.5 Combination of different practices

As observed, IndiSoft combined good practices from different approaches. Outsourcing projects could not be conducted merely by heuristic approach as described by Agile Development. They should mix between the formal and agile way of development. On the one hand, the organization maintained the traditional and conservative methodology (waterfall), while on the other hand, they internally adopted agile practices such as creating the "ecosystem" (cubicle, ip messenger, information osmosis, etc). The process dimension was formal (waterfall), but the the people dimension was agile(dynamic assignment).

The organization also established a process which combined the external standard (ISO 9001) and the methodology (waterfall) as an answer to the development challenges such as variation in people and variation across projects. Recognizing the variety of projects they established two practice groups (SmartView and SmartBiz) and applied the process accordingly. They managed to draw a balance between discipline, skills, and understanding – the elements which are more emphasized in Agile Development – and process, formality, and documentation. In fact, these cases showed that QMS played a more central role in determining the success of an outsourcing project than methodology per se.

Chapter 8

Conclusion

In this chapter, key messages and contributions of the study are presented, grouped by the various targeted audience. This thesis provides contribution for various audience: IS researchers, methodology authors and software outsourcing practitioners.

To IS Researchers

I enhance the notion of quality in software development and emphasize its importance in the context of GSW. Although quality has been taken for granted as an emerging outcome of a disciplined process, here it is seen in a new and different way. Drawing from a richer view of software as a multidimensional entity (product, process, people), quality can be seen not just as a monolithic concept relying merely on the process view, but as one consisting of *invariant* elements derived from common practices and *variable* elements derived from negotiation between all actors involved in the development process. This view of quality as well as the contemporary approaches to it (software methodologies and QMS) are integrated in a framework which can be useful to monitor, identify and analyse quality issues in software projects.

To Methodology Authors

Using the multidimensional view of software, methodology authors can devise a more comprehensive strategy to develop software by addressing various issues of quality more closely. They are expected to be more aware of the variable elements of quality and design the methodology accordingly. And most importantly, drawing from the experience of distributed

(as opposed to co-located) development, they can include a strategy to resolve the inherent challenges posed by the distibuted context.

To GSO Practitioners

This thesis recommends best practices for GSO practitioners which are drawn from succesful experiences in the case studies while recognizing the importance of people dimension of software – i.e. customers – to achieve such a success.

Bibliography

[Ahern et al., 2003] Ahern, D. M., Clouse, A., and Turner, R. (2003). *CMMI Distilled: A Practical Introduction to Integrated Process Improvement*. Addison Wesley, 2nd edition.

[Avison and Fitzgerald, 2003a] Avison, D. and Fitzgerald, G. (2003a). *Information systems development : methodologies, techniques and tools*. McGraw-Hill, London. TY - BOOK.

[Avison and Fitzgerald, 2003b] Avison, D. E. and Fitzgerald, G. (2003b). Where now for development methodologies? *Commun. ACM*, 46(1):78–82.

[Barret et al., 1997] Barret, M., Sahay, S., Hinings, B., and Krishna, S. (1997). The process of building gso relationships: the experience of a multinational vendor with indian contractors. In *ICIS '97: Proceedings of the eighteenth international conference on Information systems*, pages 500–501, Atlanta, GA, USA. Association for Information Systems.

[Binder, 2000] Binder, R. (2000). *Testing Object-Oriented Systems*. Addison-Wesley.

[Brooks, 1987] Brooks, F. P. (1987). No silver bullets: Essences and accidents of software engineering. *IEEE Computer*, April 1987:10 – 19.

[Brooks, 1995] Brooks, F. P. (1995). *The mythical man-month : essays on software engineering*. Addison-Wesley, Reading, Mass. TY - BOOK.

[Bruegge and Dutoit, 2004] Bruegge, B. and Dutoit, A. (2004). *Object Oriented Software Engineering Using UML, Pattern and Java*. Pearson Prentice Hall.

[Bryant, 2000] Bryant, A. (2000). It's engineering jim , but not as we know it: software engineering – solution to the software crisis, or part of the

problem? In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 78–87, New York, NY, USA. ACM Press.

[BusinessWeek, 2005] BusinessWeek (2005). Cafes, beaches, and call centres. *BusinessWeek 5/12 September*, page 23.

[Buzan, 1995] Buzan, T. (1995). *The mind map book*. BBC, London. TY - BOOK.

[Buzan, 2004] Buzan, T. (2004). www.mind-map.com.

[Castells, 2000] Castells, M. (2000). *The rise of the network society*, volume 1 of *The Information Age: Economy, Society and Culture*. Blackwell Publishers Ltd, Oxford. TY - BOOK.

[Cockburn, 2001] Cockburn, A. (2001). *Agile software development*. Addison-Wesley, Boston, Mass. TY - BOOK.

[Cockburn, 2003] Cockburn, A. (2003). *People and methodologies in software development*. Faculty of Mathematics and Natural Sciences, University of Oslo, Oslo. TY - BOOK Delvis opptrykk av artikler.

[Costlow, 2003] Costlow, T. (2003). Globalization drives changes in software careers. *Software, IEEE*, 20(6):14–16. TY - JOUR.

[Crosby, 1979] Crosby, P. B. (1979). *Quality is free : the art of making quality certain*. McGraw-Hill, New York.

[Crowther and Clarke, 2005] Crowther, D. C. and Clarke, P. J. (2005). Examining software testing tools. *Dr. Dobb's Journal*, #373 June(373):26 – 33.

[Cusumano et al., 2003] Cusumano, M., MacCormack, A., Kemerer, C., and Crandall, B. (2003). Software development worldwide: the state of the practice. *Software, IEEE*, 20(6):28–34. TY - JOUR.

[Das et al., 1999] Das, A., Soh, C., and Lee, P. (1999). Client satisfaction with outsourced it services: a transaction-cost approach. In *Proceeding of the 20th international conference on Information Systems*, pages 518–523. Association for Information Systems, Charlotte, North Carolina, United States.

[Dibling, 2005] Dibling, J. (2005). Debugging production software. *Dr. Dobb's Journal*, #373 June:42–46.

[Eickelman, 2003] Eickelman, N. (2003). An insider's view of cmm level 5. *Ieee Software*, 20(4):79–81.

[Fairchild, 2004] Fairchild, A. (2004). Information technology outsourcing (ito) governance: an examination of the outsourcing management maturity model. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 232–239. TY - CONF.

[Feller and Fitzgerald, 2000] Feller, J. and Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. In *ICIS '00: Proceedings of the twenty first international conference on Information systems*, pages 58–69, Atlanta, GA, USA. Association for Information Systems.

[Filman et al., 2004] Filman, R. E., Elrad, T., Clarke, S., and Aksit, M. (2004). *Aspect-Oriented Software Development*. Addison Wesley Professional.

[Frühauf, 1994] Frühauf, K. (1994). Software quality: Concern for people. In *4th European Conference on Software Quality*, VDF, Zurich.

[Garvin, 1984] Garvin, D. (1984). What does product quality mean? *Sloan Management Review*, 4.

[Geertz, 1973] Geertz, C. (1973). *The interpretation of cultures : selected essays*. Basic Books, New York. TY - BOOK.

[Gillies, 1997] Gillies, A. (1997). *Software quality: theory and management*. International Thomson Computer Press, London. TY - BOOK.

[Haag et al., 1996] Haag, S., Raja, M. K., and Schkade, L. L. (1996). Quality function deployment usage in software development. *Commun. ACM*, 39(1):41–49.

[Heeks et al., 2001] Heeks, R., Krishna, S., Nicholson, B., and Sahay, S. (2001). Synching or sinking: Global software outsourcing relationships. *IEEE Software*, 18(2):54–60.

[Holmquist, 2005] Holmquist, L. E. (2005). Prototyping: generating ideas or cargo cult designs? *interactions*, 12(2):48–54.

[Hosny, 2004] Hosny, H. (2004). The cmm software quality assurance process scaled down for small organizations. In *Electrical, Electronic and Computer Engineering, 2004. ICEEC '04. 2004 International Conference on*, pages 291–294. TY - CONF.

[Huo et al., 2004] Huo, M., Verner, J., Zhu, L., and Babar, M. (2004). Software quality and agile methods. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pages 520–525 vol.1. TY - CONF.

[Imsland, 2003] Imsland, V. (2003). *The role of trust in global software outsourcing relationships*. V. Imsland, Oslo. TY - BOOK Hovedoppgave i informatikk (Cand.scient.) - Universitetet i Oslo, 2003.

[Imsland and Sahay, 2005] Imsland, V. and Sahay, S. (2005). 'negotiating knowledge': The case of a russian-norwegian software outsourcing project. *Scand. J. Inf. Syst.*, 17(1):101–130.

[Jackson, 2004] Jackson, M. (2004). Seeing more of the world [requirements engineering]. *Software, IEEE*, 21(6):83–85. TY - JOUR.

[Khoshgoftaar et al., 2004] Khoshgoftaar, T., Liu, Y., and Seliya, N. (2004). A multiobjective module-order model for software quality enhancement. *Evolutionary Computation, IEEE Transactions on*, 8(6):593–608. TY - JOUR.

[Kitchenham, 1989] Kitchenham, B. A. (1989). Software quality assurance. *Microprocess. Microsyst.*, 13(6):373–381.

[Klein and Myers, 1999] Klein, H. K. and Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Q.*, 23(1):67–93.

[Krishnan, 1993] Krishnan, M. S. (1993). Cost, quality and user satisfaction of software products: an empirical analysis. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1*, pages 400–411. IBM Press, Toronto, Ontario, Canada.

[La Ferla, 2004] La Ferla, B. (2004). Offshore outsourcing: out of favour? *IEE Review*, 50(3):26–27. TY - JOUR.

[Laplante et al., 2004] Laplante, P., Costello, T., Singh, P., Bindiganavile, S., and Landon, M. (2004). The who, what, why, where, and when of it outsourcing. *IT Professional*, 6(1):19–23. TY - JOUR.

[Lichter et al., 1993] Lichter, H., Schneider-Hufschmidt, M., and Zllighoven, H. (1993). Prototyping in industrial software projects – bridging the gap between theory and practice. In *Proceedings of the 15th in-*

ternational conference on Software Engineering, pages 221–229. IEEE Computer Society Press, Baltimore, Maryland, United States.

[Malloy and Voas, 2004] Malloy, B. and Voas, J. (2004). Programming with assertions: a prospectus [software development]. *IT Professional*, 6(5):53–59. TY - JOUR.

[McLaughlin, 2003] McLaughlin, L. (2003). An eye on india: outsourcing debate continues. *Software, IEEE*, 20(3):114–117. TY - JOUR.

[Mich et al., 2003] Mich, L., Franch, M., and Gaio, L. (2003). Evaluating and designing web site quality. *IEEE Multimedia*, 10(1):34–43.

[Mintzberg, 1983] Mintzberg, H. (1983). *Structure in fives : designing effective organizations*. Prentice-Hall, Englewood Cliffs, N.J.

[Mockus et al., 2005] Mockus, A., Zhang, P., and Li, P. L. (2005). Predictors of customer perceived software quality. In *Proceedings of the 27th international conference on Software engineering*, pages 225–233. ACM Press, St. Louis, MO, USA.

[Natt och Dag et al., 2005] Natt och Dag, J., Regnell, B., Gervasi, V., and Brinkkemper, S. (2005). A linguistic-engineering approach to large-scale requirements management. *Software, IEEE*, 22(1):32–39. TY - JOUR.

[Pressman, 2005] Pressman, R. S. (2005). *Software engineering : a practitioner's approach*. McGraw-Hill Higher Education, Boston.

[Pyzdek, 2003] Pyzdek, T. (2003). *Quality Engineering Handbook, 2nd Ed*. Marcel Dekker, Inc., New York - Basel, 2nd edition.

[Robertson, 1995] Robertson, S. (1995). Visibility: the key to quality improvement. *Software, IEEE*, 12(4):95–97. TY - JOUR.

[Sahay, 2003] Sahay, S. (2003). Global software alliances: the challenge of 'standardization'. *Scand. J. Inf. Syst.*, 15(1):3–21.

[Sahay et al., 2003] Sahay, S., Nicholson, B., and Krishna, S. (2003). *Global IT outsourcing: software development across borders*. Cambridge University Press, Cambridge.

[Sharp et al., 2000] Sharp, H., Robinson, H., and Woodman, M. (2000). Software engineering: community and culture. *Software, IEEE*, 17(1):40–47. TY - JOUR.

[Siakas, 2002] Siakas, K. (2002). What has culture to do with spi? In *Euromicro Conference, 2002. Proceedings. 28th*, pages 376–381. TY - CONF.

[Sloper, 2004] Sloper, A. (2004). Meeting the challenge of outsourcing. *Engineering Management Journal*, 14(3):34–37. TY - JOUR.

[Sommerville, 2004] Sommerville, I. (2004). *Software engineering*. Pearson/Addison-Wesley, Boston.

[Tiwana, 2004] Tiwana, A. (2004). Beyond the black box: knowledge overlaps in software outsourcing. *Software, IEEE*, 21(5):51–58. TY - JOUR.

[Vaughan-Nichols, 2003] Vaughan-Nichols, S. (2003). Building better software with better tools. *IEEE JNL Computer*, 36(9):12–14.

[Voas, 2004] Voas, J. (2004). Software's secret sauce: the -ilities [software quality]. *Software, IEEE*, 21(6):14–15. TY - JOUR.

[Walsham, 1993] Walsham, G. (1993). *Interpreting information systems in organizations*. Wiley, Chichester. TY - BOOK.

[Ward and Aurum, 2004] Ward, J. and Aurum, A. (2004). Knowledge management in software engineering - describing the process. In *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pages 137–146. TY - CONF.

[Wood, 2003] Wood, A. P. (2003). Software reliability from the customer view. *IEEE JNL Computer*, 36(8):37–42.

[Yilmaz et al., 2004] Yilmaz, C., Memon, A., Porter, A., Krishna, A., Schmidt, D., Gokhale, A., and Natarajan, B. (2004). Preserving distributed systems critical properties: a model-driven approach. *Software, IEEE*, 21(6):32–40. TY - JOUR.

Glossary of Acronyms

- CMM: Capability Maturity Model
- CMMI: Capability Maturity Model Integration
- DLC: Development Life Cycle
- GSO: Global Software Outsourcing
- GSW: Global Software Work
- ICT: Information & Communication Technology
- ISO: International Organization for Standardization
- MNC: Multi National Company
- PL: Project Leader
- PM: Project Manager
- QFD: Quality Function Deployment
- QMS: Quality Management System
- SDLC: Software Development Life Cycle
- SPICE: Software Process Improvement and Capability dEtermination
- TCE: Transaction Cost Economics
- TM: Team Member
- TQM: Total Quality Management
- V&V: Verification & Validation