Reflective Steps: A Collaborative Learning Oriented Approach to Software Development and Process Improvement

Dissertationsschrift zur Erlangung des Grades eines
Doktors der Naturwissenschaften
am Department Informatik
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg

Vorgelegt von
Tesfaye Biru
aus Addis Abeba, Äthiopien

Okobter 2008

Betreut von:
Prof. Dr. Christaine Floyd, Universität Hamburg

Gutachter/innen:

Professor Dr. Christiane Floyd (Universität Hamburg)
Professor Dr. Ingrid Schirmer (Universität Hamburg)
Professor Dr. Gustav Pomberger (Johannes Kepler Universität, Linz)

Tag der Disputation: 17 Oktober, 2008

Dedicated to

My postgraduate students and fellow software practitioners

ACKNOWLEDGMENT

" If I have seen further, it is by standing on the shoulders of others" (Sir Isaac Newton)

As with any piece of research undertaking, in the course of conducting this piece of intellectual research, I have been assisted (in one form or another) by many people. I would like to deeply thank those without whom this study would not have matured.

This research gave me a unique and challenging opportunity, as well as the privilege, to meet and work with giants and paradigm pioneers in the field of software engineering. In this connection, first and foremost, I would like to thank Professor Dr. Christiane Floyd for challenging me to start a PhD research and providing me with required guidance for successful completion. I would like to gratefully acknowledge the stimulating, insightful and resourceful discussions and conversations we had in the course of this work both during my stays in Hamburg and her visits in Addis. Particularly, her insight on scholarship and methodical introspection, as well as her instinct for kindness, charity and love had been very inspirational and supportive through and beyond the duration of the research. Likewise, I am deeply thankful to my reviewers, Professor Dr. Ingrid Schirmer and Professor Dr. Gustav Pomberger for accepting and critically reviewing my thesis on top of their busy academic schedules. I am also extremely grateful for their clarifying criticisms, encouragement and comforting treatment during and after the defense.

My special thanks to Professor Wolfgang Menzel for accepting to be the Head of the PhD Committee and to all at the University of Hamburg who made arrangements for the defense to take place in the form of Video conferencing.

I would like to thank the Ministry of Capacity Building at Addis Ababa for financing my visits to Hamburg. I am extremely grateful to the Management of Addis Ababa University, and Civil Service College of Ethiopia, for understanding the difficult situation I was in and facilitating special arrangement to enable me to defend my work through video conferencing.

My debt to my wife, professional colleague and partner Dr. Rahel Bekele is beyond measure. Thank you Rahel for spending considerable time reading, commenting and editing the manuscript. Thank you for standing by me, giving me love, encouragement, and bearing patiently with me and the family throughout the challenging and difficult times. Thank you for making difficult things easy every step of the way, for being there for me and my responsibilities, no matter what. I just cannot thank you enough.

My software development experience has been rewarding because of the caliber of people with whom I have been fortunate enough to work with. In this connection, I owe my gratitude to all my former excellent postgraduate students (now professional colleagues) and to Ato Solomon Yewondwossen and Ato Mebrahtu H/Mariam.

To Dr. Hailu Ayele, Professor Nigussie Tebeje, Professor Mogessie Ashenafi, Professor Eshetu Wencheko, Dr. Tetemke Mehari, Professor Sebsibie Demissew and Ato Mulushewa Mulat, whose friendships and advice, I have come to value most. Thank you for your ever comforting, intimate and encouraging support.

To Aida Bahta, Martha Yifiru, Berhanu Beyene and Solomon Teferra thank you for your hospitality and kindness during my extended stay in Hamburg.

My beloved children Tibebu, Elshadai, Tehut and Bezawit, thank you for your love, for understanding my absence from home during repeated trips abroad, lost weekends, and odd working hours.

To Tsehay Gashaw and Tigist Gebru, thank you for your motivation, encouragement and love when it was most required.

I am forever indebted to my mother Geremush Bogale for instilling an ethic of hard work, courage, endurance and educational achievement in me, and also for her enduring love.

Tesfaye Biru

October, 2008

ABSTRACT

This research attempted to explore the possibility of using context sensitive methodical approaches to address the software development challenges in Ethiopia. Based on extensive case studies and surveys supported by reflections on the researcher's years of practical experience in both teaching and practicing software development, the situation in Ethiopia including the challenges faced by practitioners were documented. According to the findings, the software development situation is mostly dominated by failure cases characterized by: unrealized benefits, unsatisfied users, substantial budget and time overruns far beyond expected, frustrated developers, etc.

Among the main causes identified for the failures is the oversize gap between demand and supply. On the demand side, most outsourced projects: are very large (by local standards); involve the development of multiple applications systems for specific organizations; involve business process redesign as a front-end process to the software development; and operate in unstable organizational environments. On the supply side, most of the software development firms: are inexperienced and small; follow ad hoc processes and methods; lack competence in project management and soft-skills; and are affected by very high staff turnovers. There are inadequate educational and training support infrastructure and absence of home-grown or contextualized methods, as well as absence of national standards or guidelines.

In this research, methodical approaches that address contextual issues on both demand and supply sides are considered to tackle the gap. On the demand side, strategies around project design that involve scoping, prioritizing, outsourcing and the like were considered. On the supply side, competence development measures, at both organizational level (software process improvement) and individual level (learning and training mechanisms), were considered. Most importantly, institutionalization of collaborative approaches between developers and users, based on transformational participation principles, and feedback-based reflective learning, were considered.

The effort resulted in the development of a comprehensive methodical approach known as Reflective Steps. The approach evolved from the process of tailoring the STEPS model and complementary aspects of contemporary methods and processes based on the contextual issues identified. In the process, home-grown collaborative techniques and proven practices in the areas of business process redesign for software development and project management were incorporated.

As proposed, Reflective Steps provides an explicit treatment of software development activities which are considered important to the local setting but either missing or implicitly treated in popular methods and processes. An integrated development cycle that combines project design, application production, and application embedment and use aspects is proposed. Reflective Steps promotes the process of discovering suitable processes and methods for a project in the course of developing the software itself. As such, it uses a contextualized approach as a starter and proceeds with a step-by-step improvisation process through collaborative reflective learning based project experience. For this purpose, a multi-level collaborative reflective learning model is introduced by tailoring organizational learning and communication models. Combination of single-loop learning and double-loop learning based on reflections on process, progress, product and context are proposed. Each learning cycle involved action-reflection-improvement. A Reflective Steps workshop technique is also developed as a learning platform.

Although the approach for the most part evolved from years of experience in teaching and practice, attempts were also made as part of this research to further experiment with aspects of Reflective Steps in real-life project environments. Encouraging results were obtained in the field experiments conducted both in software development project and in teaching at postgraduate studies. In particular, the experiment in the teaching area showed promising results and optimism on how Reflective Steps could be used to better educate students with practical skills through the integration of real-life problem scenarios into software engineering curriculum. Taken together, with further work, it is expected that Reflective Steps will gradually achieve wider acceptance and contribute to increased use of methodical approaches that would in turn contribute to the improvement in the productivity of project teams and quality of products in the local setting.

ZUSAMMENFASSUNG

Diese Forschungsarbeit stellt einen Versuch dar, kontextsensitive methodische Ansätze zu nutzen, um den Herausforderungen der Softwareentwicklung in Äthiopien zu begegnen. Auf der Grundlage von Langzeit-Fallstudien und Umfragen, untermauert durch Reflexion über die langjährigen praktischen Erfahrungen des Forschers in Lehre und Praxis der Softwareentwicklung, wird die Situation in Äthiopien und die Herausforderungen für die Praktiker dokumentiert. Die Untersuchung macht deutlich, dass die Lage der Softwareentwicklung vorwiegend durch das Scheitern von Projekten dominiert wird. Kennzeichnend sind: nicht eingetretene Vorteile, unzufriedene Benutzer, substantielles Überschreiten von Budget und Zeit weit über alle Erwartungen, frustrierte Entwickler etc.

Als einer der wichtigsten Gründe für das Scheitern wurde die übergroße Lücke zwischen Nachfrage und Angebot identifiziert. Auf der Nachfrageseite sind die meisten in Auftrag gegebenen Projekte sehr groß (für lokale Verhältnisse). Sie beinhalten die Mehrfach-Entwicklung von Anwendungssystemen, die auf spezielle Organisationen zugeschnitten sind, und die Neugestaltung von Geschäftsprozessen im Vorfeld der Softwareentwicklung. Die Systeme werden in einem instabilen Organisationsumfeld betrieben. Auf der Angebotsseite sind die meisten Softwarefirmen unerfahren und klein; sie befolgen Ad-Hoc-Prozesse und -Methoden, haben wenig Kompetenz im Management und in Soft-Skills, und leiden unter einer hohen Personalfluktuationsrate. Die Ausbildungs- und Trainingsinfrastruktur zu ihrer Unterstützung ist inadäquat, selbst entwickelte oder kontextualisierte Methoden fehlen ebenso wie nationale Standards und Richtlinien.

Um die Lücke zu überwinden, werden in dieser Arbeit methodische Ansätze behandelt, die Kontextfragen auf der Nachfrage- und der Angebotsseite adressieren. Auf der Nachfrageseite geht es um Strategien im Umfeld des Projekt-Design: um das Abgrenzen von Projekten, das Setzen von Prioritäten, die Vergabe von Aufträgen und ähnliches. Auf der Angebotsseite werden Maßnahmen zur Kompetenzentwicklung auf der Organisationsebene (Software Process Improvement) und der individuellen Ebene (Lern-

und Trainingsansätze) behandelt. Vor allem wird der Institutionalisierung von Ansätzen zur Zusammenarbeit von EntwicklerInnen und NutzerInnen auf der Grundlage von transformationsorientierten partizipativen Prinzipien und reflektivem Lernen in Rückkopplungszyklen eine hohe Bedeutung zugemessen.

Ergebnis der Forschung ist die Entwicklung eines umfassenden Methodenansatzes unter dem Namen *Reflective Steps*. Dieser Ansatz hat seinen Ursprung im STEPS-Modell, das angepasst und um komplementäre Aspekte von aktuellen Methoden und Prozessen auf der Grundlage der identifizierten kontextuellen Anliegen angereichert wurde. In diesem Prozess wurden selbst entwickelte Techniken der Zusammenarbeit und bewährte Praktiken des Geschäftsprozessentwurfs für Softwareentwicklung und Projektmanagement einbezogen.

Reflective Steps bietet eine explizite Behandlung von Aktivitäten der Softwareentwicklung, die im lokalen Kontext bedeutsam erscheinen, aber in verbreiteten Methoden und Prozessen entweder fehlen oder nur implizit behandelt werden. Ein integrierter Entwicklungszyklus wird vorgeschlagen, der Projekt-Design, die Entwicklung und Einbettung von Anwendungen sowie Aspekte der Nutzung umfasst. Reflective Steps unterstützt den Prozess, geeignete Vorgehensweisen und Methoden für ein Projekt im Verlauf der Softwareentwicklung selbst herauszufinden. Als Ausgangpunkt dafür verwendet es einen kontextualisierten Ansatz und setzt sich fort in kollaborativem, reflektivem Lernen auf der Basis von Projekterfahrungen. Zu diesem Zweck werden Ansätze zum Verständnis von Lernen und Kommunikation in Organisationen angepasst, um ein Modell für kollaboratives, reflektives Lernen auf mehreren Ebenen einzuführen. Vorgeschlagen wird die Kombination von Lernen in einfachen und doppelten Rückkopplungsschleifen auf der Grundlage von Reflexion über Prozess, Fortschritt, Produkt und Kontext. Jeder Lernzyklus beinhaltet Aktion-Reflexion-Verbesserung. Als Forum für kollaboratives Lernen wird eine Reflective Steps Workshop-Technik vorgestellt.

Obwohl sich der Ansatz in großen Teilen auf langjährige Erfahrung in Lehre und Praxis gründet, wurden im Rahmen dieser Forschung auch weitere Experimente mit Aspekten

von Reflective Steps durchgeführt. In der experimentellen Anwendung bei Softwareentwicklungsprojekten und der postgraduierten Lehre wurden ermutigende Ergebnisse erzielt. Insbesondere führten die Experimente im Lehrbereich zu viel versprechenden Ergebnissen und zu optimistischen Einschätzungen darüber, wie Reflective Steps genutzt werden könnte, um echte Problem-Szenarien ins Software Engineering einzubeziehen und Studierende praxisgerechter auszubilden.

Insgesamt ist zu erwarten, dass Reflective Steps nach entsprechender Weiterentwicklung allmählich eine breitere Akzeptanz gewinnen und zur vermehrten Anwendung von Methoden beitragen wird, welche die Verbesserung der Produktivität von Projektteams und die Qualität der Produkte im lokalen Kontext erhöhen.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

AAU	Addis Ababa University
AC	Abstract Conceptualization
AE	Active Experimentation
CCB	Customer Care and Billing
CE	Concrete Experience
CIBAS	Complete Insurance Business Application Software
CMM	Capability Maturity Model
DLL	Double Loop Learning
EICTDA	Ethiopian ICT Development Agency
ESTC	Ethiopian Science and Technology Commission
ETC	Ethiopian Telecommunication Corporation
ETHICS	Effective Technical and Human Implementation of Computer Systems
IBAS	Integrated Business Application Software
IFMS	Integrated Financial Management System
JAD	Joint Application Development
NCC	National Computer Center
NGPM	Next Generation Process Model
PMBOK	Project Management Body of Knowledge
RO	Reflective Observation
RPG	Report Program Generator
RSD	Reflective System Development
RUP	Rational Unified Process
SLL	Single Loop Learning
SSM	Soft Systems Methodology
STEPS	Software Technology for Evolutionary Participative System Development
TIN	Tax Identification Number
TOR	Terms of Reference
UCC	United Computer Consultants
UNIC	United Insurance Company

CHAPTER ONE

1. Setting the Scene

The overall purpose of this research is to explore the possibility of developing suitable approaches to address the software development challenges in Ethiopia by tailoring publicly available methods and process models.

In today's dynamic and competitive business environment, software has become a critical organizational resource and economic commodity. On the other hand, the development of large software systems has become a rather complex business which is encumbered by many problems (Pomberger, 2006). Given such complexity, although not considered as a panacea for all related problems, there is a general acknowledgement that the use of some kind of methods to guide the process of development would help in tackling aspects of these problems. To this end, we see now many methods and process models that are developed by the software engineering community and made publicly available for use. These methods range from those traditional prescriptive/rigid and technically oriented approaches (often criticised for not being sensitive to project contexts and the human dimension) to the more recent people-oriented and participatory approaches proclaimed to address the perceived as well as acknowledged deficiencies of the traditional ones.

Most of the publicly available software methods and process models have originated from, and are being intensively used in the contexts of, developed countries (Korpela, 2001; Korpela, 1998). As such, they may not be effective if applied quite literally to local situations. Needless to say, the differences between the environmental/contextual as well as project situations in these countries and that of Ethiopia are too obvious. There are differences in application requirements, jobs and work environments, attitudes and behaviours in the workplaces, organizational structures, etc.

On the other hand, system development methods and process models that have either originated from, or been customized for use in, the local environment are generally lacking. What is more, the extent to which the publicly available approaches (developed and used elsewhere) lend themselves to adjustability to fit into the local contexts has not been adequately explored. In fact, very little is generally known about the local software development situation.

Within the foregoing as a framework, this chapter attempts to set the scene for this work. It starts by reviewing related literature on software development approaches. This is followed by summarizing the software development situation in Ethiopia. With these as background, the research questions and the research approach followed to address the research questions are introduced. This is followed by the presentation of the summary of findings and results from the research endeavor. The last section describes the organization of the research report.

1.1 Software Development Approaches

The evolution of the work on the design of systems development approaches[1] over the years has concurrently proceeded in two dimensions: process models and methods. According to tradition (Boehm, 1988), while the process model dimension concerns itself with naming, describing and sequencing of activities involved in a typical software development project, the methods dimension deals with the techniques of performing the activities identified in the process models. In published literature, while the former is often discussed under such themes as phased/linear and/or cyclic/iterative, descriptive and/or prescriptive, etc., the latter is discussed under such software modeling and specification themes as formal approaches, structured and object-oriented approaches. What is more, it is not uncommon to find these two dimensions mixed with each other as exemplified by: Pomberger's Prototyping (as an activity and a technique) (Pomberger and Blaschek, 1996 as cited in Pomberger, 2006), Boehm's WinWin Spiral (Boehm, 1998) which introduced methods within the original process model, and object-oriented analysis and design approaches. In general terms, there is abundant literature published on these issues over the years. Many research investigations and numerous software

[1] To avoid unnecessary confusion, throughout this work the terms 'approach' and 'method', when used in the context of 'software approach' and 'software method' respectively, are used interchangeably to refer to both methods and processes combined – in other words, to refer to any collection of models, techniques and tools which help to make software development more systematic. The use of the term "methodology" is deliberately avoided unless otherwise felt necessary (when it is used, it connotes the "study of method"). The term 'methodologist' is however used to refer to workers on methods and process models.

engineering guidebooks compare and contrast these issues. Readable accounts can be found, for instance in (Austin and Paulish, 1994; Mcdermid, 1993; Green and DiCaterion, 1998; Boehm, 1988; Pomberger, 2006; Wieringa 1998; Loy, 1990; Scacchi, 2002; Abrahamsson, 2002).

On the methods front, closer examination of published literature (Austin and Paulish, 1994; Curtis, 1992:83) reveals that much of the earlier work was focused heavily on standardizing format and specifying aspects of software development. Serious attempts were made to address technical problems of creating standard languages (textual, graphical or mathematical) and technical artifacts that permit the use of unambiguous representation and specification of processes that can be rigorously, and perhaps automatically, verified. These are exemplified by the various formal languages, CASE tools, UML, reusable software components, etc.

As formalizations were developed to an extent, new challenges from the one-sided focus on formalization-related methods discourse emerged.

> "Despite some progress in the development of more powerful tools and mathematically based specification techniques, the results have often been less promising than expected. Still, the quality of software is only revealed to its full extent once it is in use. Software projects fail to live up to the expectations of developers and managers or of the domain experts who ultimately have to use the product" (Keil-Slawik, 1992: 168).

> "A software development process can not be fully formalized because it is a social process: human/task/technology systems are developed by people for people, and that demands high social competence and team work, which can not be fully formalized" (Pomberger, 2006).

As a result, a new set of demands emerged: more involvement of stakeholders in the development process to enhance quality and increase level of use, flexibility in handling changing requirements, better speed in the delivery of products, and the inclusion of measures to determine risks and effectiveness, among others. In this context, organizational and communication problems became critical. Problems related to the human aspect of making appropriate use of process representations, to make them helpful tools rather than bureaucratic obstacles, required more attention.

In this connection Floyd (1992:25) wrote,

> "software developers get little guidance for understanding the use-situation, where people are carrying out their work with the help of the computer. … An adequate consideration of the embedding of computer programs in the human world does indeed require us to go scientifically beyond the formal and mathematical methods provided for in traditional computer science, and to open ourselves to approaches from the humanities."

For this purpose, Floyd argued, by starting with the already developed approaches elsewhere for understanding human learning and communication, individual and cooperative work, and the interrelationship between technology and organizations, the software engineering community must face the task of tailoring suitable approaches to the needs of the software engineering discipline.

As Rauterberg and Strohm (1992) noted, one of the principal problems of traditional software development lies in the fact that those technical people who have been primarily involved in software development to date have not been willing to recognize that software development is, in most cases, mainly a question of task, job and/or organizational planning. To address this problem effectively, "we must start learning to plan jointly technology, organization and the application of human qualification.   .. Technology should be viewed as one way of providing the opportunity to organize our living and working environments in a manner which is better suited to human needs." (Rauterberg and Strohm, 1992:128).

According to Bjerknes et al. (1990), any system development project itself is an organization. It is a collective undertaking that involves many persons and groups requiring cooperation, timeliness and management. To be successful, such projects (particularly the large ones) have to create and maintain temporary organizational networks linking users to system developers, decision makers to workers, and consultants to clients. Therefore, in dealing with such applications, both technical and organizational competences are key factors.

For these reasons, the human and social roles in software development, an area that was not explored enough by the software engineering community traditionally, started to form one of the central topics in software engineering. Today, organizational issues and such social aspects as cooperative, participative, learning-oriented and adaptive working techniques are among the most explored in the field of software engineering to meet the challenge from the industry and users (Floyd, 1987; Mathiassen and Nielsen, 1989; Floyd, 1992; Boehm and Bose, 1994; Pomberger, 2006; Cockburn, 2006;).

Likewise on the process model front, efforts to bring control and discipline to what had previously been a rather unstructured and chaotic process, resulted in the introduction of systematic approaches. Guided by traditional engineering practices, the earlier versions of systematic approaches introduced structured, linear time-delineated stage models and defined milestones in software development process that included: problem/requirements analysis, conception, specification and planning, programming, test and implementation, and operation and maintenance (Pomberger, 2006; Rauterberg and Strohm, 1992). Approaches based on such a setup are commonly called 'traditional'. Alternative labeling conventions used in the literature include product-oriented (Floyd, 1987) or phase-oriented (Pomberger, 2006) or plan-driven or document-driven (Boehm, 1988). As the names suggest, a common feature for the traditional approach is its emphasis on defining the phases, scope, schedule, and costs of the project upfront including, for instance, an early fixing stage and extensive documentation of the end product requirements, and thereafter executing on the specifications in an efficient manner. Development stages are performed sequentially, with reviews at the end of each stage ensuring that all necessary work has been completed to that point. Here, the system developer mostly is a technological expert enacting the story of modernism; and attempts are made to address the problem with the developers own 'mental construct' or interpretive scheme - user participation is minimal. One popular example of such traditional process model is the Waterfall Model.

While these traditional approaches proved to be successful responses to the early problems that had plagued software development, they are generally viewed critically and their validity in today's dynamic environment is questioned, because they have the following major drawbacks (Floyd, 1987; Pomberger, 2006; Cockburn, 2006):

- The phases and activities are primarily oriented to software engineering aspects rather than the requirements of the application domain.
- The purely sequential approach proves impossible to adhere to and difficult to plan reasonably.
- Requirements can be defined only partially in advance and change constantly.
- Pure documents produced at each milestone do not provide reliable intermediate results as they lack sufficient meaning.
- The one-sided emphasis on formalization at the expense of communication, learning and evolution ignores the participants' learning potential and fails to facilitate cooperation between developers and users during development.
- There is no systematic feedback about design from the participants.
- These approaches are prescriptive and rigid as they emphasise laying down standardized working procedures to be followed without reference to the specific project situation at hand (do not provide for the flexibility required in practice).
- These approaches fail to take into account the quest for quality (in the sense of end-use) and neglect any sort of foundation for human-oriented system design.
- These approaches assume that problems to be addressed by software development are well defined and objective, and the development and use take place in a static environment.
- Production of software cannot be separated from use and maintenance.

These serious limitations of the traditional approaches have led to proposals of alternative approaches that try to address both acknowledged and perceived drawbacks of the traditional approaches. In most of the contemporary approaches, the software development process is generally understood to be evolutionary, typically involving iterative cycles of: design and prototyping, implementation, evaluation and revision for the purpose of delivering the required software incrementally. The approaches are more flexible rather than predictive, they emphasise participative communication & learning process, use context (workplace and application orientation), visioning and organizational embedding. Important examples include Floyd's Software Technology for Evolutionary Participative System Development (STEPS) (Floyd, 1989), Boehm's Spiral Model (Boehm, 1988), the Rational Unified Process (Kruchten, 2000), the Agile method (Cockburn, 2006; Beck, 2004), and Mathiassen's Reflective System Development (Mathiassen, 2002).

The introduction of these approaches is being discussed in the literature as a shift from product-oriented to process-oriented (Floyd, 1987), from phase-oriented to practice-oriented (Pomberger, 2006), from plan-driven or document-driven or heavy-weight to agility or lightweight (Boehm, 2002; Pomberger, 2006; Cockburn, 2006), and from hard to soft (Mathiassen, 2002, Checkland and Scholes, 1999). Boehm's Spiral model is often discussed in the literature as a risk-driven process model. While these viewpoints are discussed in more detail in Chapter Four and Chapter Five of this report, in so far as they concern the current research work, the following paragraphs summarize aspects of these viewpoints as a background to the discussions in subsequent sections and chapters.

In particular, the shift from product-oriented or phase-oriented to process-oriented or practice-oriented emphasizes the following, as clearly indicated in the STEPS, Reflective System Development (RSD) and Prototyping approaches (Floyd, 1989; Pomberger, 2006; Mathiassen, 2002).

- a move away from the traditional approach where software developers focused on specifying requirements based on descriptions produced from given perspectives and constructing a system that meet such pre-specified requirements, to a situation where they focus more on the processes of cooperation between developers and users to collaboratively interpret a given business situation, invent actions to improve it and by so doing gradually and jointly discover and develop the required software;
- a move away from the understanding of the software development process as an orderly process of planning, analysis, design, construction, to a process that involves: change process, project management, quality assurance, software process improvement, etc.

The heavy-weight and light-weight distinction describes the degree of formalization of the processes and the number of associated (intermediate) results or (intermediate) products. Thus heavy-weight process models are phase-oriented models like the Waterfall Model, while light-weight process models, also called agile process models, are flexible, weakly formalized, iterative process models like eXtreme Programming (Pomberger, 2006). The discussion on heavyweight and lightweight in the literature is also presented in terms of comparing the features and capabilities of agile methods and plan[2]-driven methods. According to Cockburn (2006), Beck (2004), Agile methods stress early and continuous delivery of valuable software over analysis and design (although these activities are not discouraged), active and continuous (face-to-face) communication between developers and customers, and welcome changing requirements. Agile methods derive much of their agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in plans. Comparing Agile and plan-driven methods, Boehm (Boehm, 2002: 64), wrote,

> "[in agile methods there is also the risk that the team will make irrecoverable architectural mistakes because of unrecognized shortfalls in its tacit knowledge. … [plan-driven methods] accept a risk that rapid change will make the plans

[2] The "plan" includes documented process procedures that involve tasks and milestone plans, and product development strategies that involve requirements, designs, and architectural plans.

obsolete or very expensive to keep up to date". On balancing agility and discipline, Boehm states, "although each approach [agile or plan-driven] has a home ground of project characteristics within which it performs very well, and much better than the other, outside each approach's home ground, a combined approach is feasible and preferable".

With respect to the hard and soft, these perspectives of system development approaches draw much on systems thinking and systems approach (Checkland and Scholes, 1999; Mathiassen, 2002). In particular, the hard systems approach emphasises clear, exact and true representations of the world. For hard systems thinkers, a system is typically a functional system, a machine with a determinate function, ordered and stable. The systems are out there, we see them (and we believe what we see), build them, change them, and improve them, by engineering. The system is analysed in terms of the functional roles played by its elements and their properties. The soft systems approach pursues the idea that there are always several, equally plausible perspectives of the world. The systems that we see in the world are based on our assumptions about the world and the experience of it. Such perspectives of the system will change if our perception of them changes, if we develop a new way of looking at them, if we experience and learn new things. The method of the soft systems approach is interpretation. One is encouraged, by this method, to consider different perspectives; the claim is that to learn about the world one needs to understand, express and debate on a variety of radically different perspectives.

Still, as a further extension of the soft systems approach, Mathiassen (1998) tried to introduce the dialectical systems approach. The dialectical systems approach is based on the idea that the world is always changing and that we cannot understand it unless we understand what change is and why it takes place. The claim of the dialectical approach is that we must think in terms of contradictions in order to understand, explain, and make possible changes. This is done through making contradictions explicit, negotiating perspectives, and learning about possible changes through intervention and action. Among the examples of approaches developed on the basis of such systems thinking are: the RSD of Mathiassen (Mathiassen, 2002), the Soft Systems Methodology (SSM) of

Checkland and Scholes (Checkland and Scholes, 1999). Another approach that emphasizes the human aspect in system development is the Effective Technical and Human Implementation of Computer Systems (ETHICS) of Mumford (1983).

From the point of view of process models and methods discussed in the preceding paragraphs, one may see that the hard systems thinking forms the basis for the more technically oriented traditional approaches, while the soft systems thinking resonates strongly with the more recent developments in the software engineering. In this connection, it is worth noting that, as compared to the more technical design-oriented approaches employed in software development, methods in this category take a socio-technical viewpoint in broader and organization-wide issues involved in the development of an information system[3] as an organizational subsystem. Underlying the systems thinking and its application to system development is the assumption that introduction of software systems into an organization is a multidimensional and intentional social and organizational change process, as the process usually results in changing the technical platform, information content and use pattern of the systems. For the purpose of this study, it is assumed that application software constitute essential components of every information system within an organization, and any reference to information system development is confined to development efforts that comprise application software as a significant part of the process.

On the whole, while the foregoing discussion on software development approaches has been brief on purpose, it was an attempt to outline how, over the years, the evolution of software approaches have developed in line with the needs of the industry and users. Finally, it would be amiss if we conclude this section without emphasizing the holes in the theoretical and conceptual foundation of the field of software engineering that have yet to be addressed.

[3] In some literature, software engineering differs from the field of information systems (IS) predominantly in the sense that the IS community takes into account the social and organizational aspects. This is partly because software engineering traditionally focused on practical means of developing software. However, as described in the foregoing, with the new challenge from the industry and users, the focus in software engineering is also changing. To this end, for the purpose of the work under reference, such a distinction between SE and SI has not been felt necessary.

Despite decades of work on software development approaches and widespread utilization of both traditional and recently developed approaches partly outlined above, there is no agreement yet among workers on the importance of traditional models and the value of recent models from either scientific or practice-oriented viewpoints (Pomberger, 2006). With respect to theoretical foundation and conceptual foundation, the literature indicates that the field of system development is still introduced and characterized as a field in an early stage of development, plagued by indeterminate/inappropriate theoretical grounding, conceptual in-exactitude and methodological disagreements (Boahane, 1999**;** Hart and Gregor, 2004). Workers in the field still continue to advocate or criticize existing approaches or to invent new approaches based on different philosophical views, perspectives on theories, modes of inquiry and methodical paradigms (often biased towards own background research or practical experience). Still unified/integrated and coherent techniques to effectively guide practice are lacking. In both professional practice and research, it is not uncommon to encounter diverse perceptions and disagreements on basic concepts and techniques. There are still so many open questions, so much more conflicting ideas and so many different methods, etc. For instance, in terms of scope, while some of the approaches cover the whole development process, others cover only part of the process such as requirements definition or engineering; there are issues of design for, with and by users; there are issues related to balancing plan/discipline and adaptivity/flexibility; there are issues related to the social quality of software and building of theory shared by a community; issues related to measurement matrix, etc.

To that end, the software engineering community more than ever is concerned with and engaged both in the improvements in existing methods and the development of new and better methods. In this connection, one emerging process-related research activity worth noting is the work in the area of software process improvement. Generally, this line of work is concerned with exploring ways and means of improving processes in practice for the purpose of increasing product quality or development team productivity, or reducing development time. As a result, there are now a number of available process improvement methods for use (see for example, Austin and Paulish, 1994). The most commonly encountered process improvement methods of interest to this research are briefly discussed in Chapter Six of this report.

1.2 Software Development Situation in Ethiopia

This section tries to summarize the software development situation in Ethiopia which is elaborated in more detail in chapters two and three of this report following the survey and case studies conducted for the purpose of this research.

1.2.1 The Context

It is apparent that sub-Saharan African countries are for long the poorest and technologically the least developed. Needless to say, they live under severe political and practical constraints, as compared even to most developing countries. The increased availability of Internet technology is, however, creating a new hope, for developing countries in general and sub-Saharan Africa in particular, to fight poverty and to catch up in terms of bridging the digital divide/gap between rich and poor countries. That they need to do much more and much faster to seize this opportunity to develop their countries is being strongly expressed (World Development Report, 1998/99).

In this connection, among the major problems cited frequently in published literature (Korpela, 2001; Mursu, 2000; World Development Report, 1998/99) and own observation (resulting from the researcher's participation in several large IT project implementations) for slow pace of development in this direction are: infrastructural deficiencies and wrong choice of technologies; shortage of financial resources particularly foreign exchange (and thus dependence on donors and vendors) to invest and support IT projects; lack of skilled personnel both in quantity and quality (further aggravated by high turnover, brain drain and extremely under resourced work environment); lack of planning and inability to manage change; over-politicized and bureaucratic decision-making process; lack of system development houses that specialize in application software development and lack of appropriate IT policy and strategy guidelines at both national and institutional levels.

However, as of recent there are some indications that some of these problems are being seriously addressed. The appreciation and recognition of the role of ICT in development among policy makers and executives in the government are increasing. Discussions are underway at various levels to design appropriate policies that help tackle the problems.

Coming back to the specific context of Ethiopia, particularly in the infrastructure front, although the introduction of telecommunication dates back to 1894, computers were introduced around 1961 (Tefferi, 1994). Initially accounting machines were introduced. These were replaced by full-fledged electronic data processing machines starting from 1965. Among the major suppliers of the time were: IBM (with its models 1421/814 followed by system 360/20), NCR (with its models 399, 499 followed by system 8200), Burroughs (with its models 1500 followed by system B80), and Hewlett-Packard (HP 3000) which came into the local market relatively late in the 1980. Among the earliest user organization were: Ethiopian Airlines (1961), Ethiopian Electric Light and Power Authority (1962), Economic Commission for Africa (1963), Central Statistics Office (1964), Ministry of Finance (1968), Ethio-Djibouti Railways (1969). Modes and means of programming ranged from using wiring panel to Report Program Generators. The COBOL language was introduced in the late 1970s and was widely used since the early 1980s.

Through 1970s and 1980s, various public and private organizations, including banks, higher learning institutions, industries in the transport and communications sector, international organizations, introduced minicomputers in their operations. University level education in the field started in the early 1980s at Addis Ababa University with the Department of Mathematics at the Faculty of Science and Department of Electrical Engineering at the Faculty of Technology. The situation dramatically changed with the emergence of microcomputer based systems and the establishment of the National Computer Center (NCC[4]) at the Ethiopian Science and Technology Commission. NCC, staffed with waves of highly educated (mostly at postgraduate level in Europe) young computer scientists, was one of the institutions that played a major role in shaping the local software industry during the 1980s and 1990s. There were attempts then to

[4] NCC was later reorganized under the Ethiopian ICT Development Agency, EICTDA.

introduce national guidelines on programming and systems development work as well as competency certification for professionals and institutions involved in the business of software development. For further historical perspectives on the introduction of computers to Ethiopia, see for example Tefferi (1994) and Sirak (1988).

Over the past few years, very encouraging developments were observed in the IT sector. Cognizant of the role of IT as both enabler and catalyst in national development, growth and competitiveness, serious measures were being taken to build capacity in the areas of ICT infrastructure, human resource, content and application. In particular, concrete actions were underway in the following areas.

- A national Government Agency, EICTDA, was established to spearhead ICT capacity building activities through the development of appropriate policies, strategies and programs to promote ICT development and utilization in the country.
- A national ICT policy was drafted together with a five-year national ICT for development action plan (2006-2010).
- Substantial investment was (and is still being) made to improve access and to upgrade telecom infrastructure. In this connection, a nation-wide fiber highway network deployment with broadband multimedia and Internet capability, rural connectivity program to provide better access to information and services to the rural community, and aggressive expansion of the fixed and mobile networks to cover all parts of the country equitably, are but some important initiatives worth citing.
- On top of the telecom networks, a number of enterprise networks with national coverage were also deployed with high speed. The nation-wide multimedia network to connect all government offices up to the district levels (WoredaNet), the broadband network that interconnects all high schools in the country (SchoolNet), the broadband network interconnecting all institutions of higher learning in the country (EthERNet), the broadband network interconnecting agricultural research centers in the country (AgriNet), and the network that

interconnects customs and tax administration offices in the country (RevenueNet) are among the flagship national projects in this direction.

- A national civil service reform program that aims at overhauling the public sector (particularly to increase efficiency, transparency and accountability of government operations) was initiated in almost all public service centers. This program involves business process redesign and introduction of result oriented performance management systems in the public service sector.
- To support the civil service program, and as part of the eGovernment initiative, application software development projects are also initiated in the various ministries.
- Related studies are also underway to facilitate the rapid development of the private ICT sector, development of e-commerce, establishment of IT Park and development of appropriate laws and legal frameworks to support ICT-enabled industry.

Based on the researcher's experience and observations made following discussions with key stakeholders in the sector, in addition to further upgrading and expanding, what remains to be a serious challenge on the infrastructure front is the maintenance of the infrastructure already deployed and provision of relevant and quality services in a sustained manner.

In light of the rapid increase in demand, fueled by the large scale national projects in the government and public sectors as well as the huge demands resulting from the IT-oriented reform initiatives in the service industry (such as telecom, airlines, banks, insurance companies, etc.), and the increase in presence of foreign business firms in response to the investment opportunities extended, currently the demands and pressures for better and improved software development and support services at the local level are increasing. Furthermore, the national ICT capacity building initiatives envision transforming the country into a preferred outsourcing destination in the software sector. On the other hand, despite such increase in investment and encouraging developments, there is very limited capacity to meet the demands and realize the investment and vision. This investment-benefit gap, which is slowing down the pace of development of the sector in the short run and which may lead to failure in realizing the desired socio-economic returns from the investments in the long run, has been recognized and decisions are being made to tackle this problem with high priority. In this effort, the software element was identified among the prime mover and determining factors to unlock the problems encountered in the areas of content and application. Under the circumstances, the realization of the national vision to be a preferred outsourcing destination in the software sector cannot be achieved unless concrete actions are taken to increase productivity and improve quality.

1.2.2 Software Development Practice

According to the assessments made for the purpose of this study (see Chapter Two and Chapter Three) and other studies made to assess the success and failure rates of software development projects locally as well as the capabilities of local software development firms (Rahel, 2004), the software environment in Ethiopia has a long way to go to reach the minimum maturity level expected to meet the demands from users and the industry. According to the findings, the software development situation so far is mostly dominated by expected improvements in business efficiency and value as a result of the software introduction that never materialized; substantial budget and time overruns far beyond expected; delivery of unfriendly and poor and thus unused quality software products; difficulty on the part of the users to effectively utilize the newly deployed software systems because of inadequate training; incomplete documentation and lack of timely and affordable maintenance support, and hence problem of sustainability, etc. What is obvious for all concerned parties (software vendors, professional practitioners, sponsors, etc.) is that under the existing circumstances, to carry out a software development project is not only difficult but failure-prone and damaging. For instance, there are cases where some software development projects which were donor driven and which depended heavily on external consultants for their development ended up in litigation resulting in both financial loss and damaging relationship between the client and the software company; others perpetuate the image of an ugly and antagonistic relationship between the stakeholders (even with the possibility of ending up in litigation); still others result in an intolerable delay and quality problem. (See Chapter Two for related details).

Furthermore, most of the projects are very large, designed with the ambition to address long standing organization-wide and nation-wide process and service efficiency problems. Accordingly, wherever they are considered, the software implementation strategies adopted seem to follow an “all or nothing” approach to system development and delivery rather than a set of step-by-step incremental improvements where functionality is improved over time. Attempts to realize this approach by trying to implement off-the-shelf solutions from elsewhere resulted in large design-reality gaps due mostly to: differences in work practices and styles, resistance to change, skill deficiencies, and lack of project management competence.

Where local developments are considered, the existing situation with regard to the use of methods may be summarized as follows (for details refer to chapters two and three of this report). So far, the local software development practice is dominated by the use of ad-hoc in-house guidelines that involve cyclical requirements gathering and programming/coding techniques. The use of industry standard or publicly available methods and process models, and the use of disciplined approach to manage the software projects are very low. Among the reasons frequently cited by practitioners in this connection are: technical skill deficiency due mainly to lack of appropriate and practical training on the methods; inappropriateness or weak-fitness of the methods to local problems and environment; absence and/or unavailability of related tools and guidelines. Software processes for both management and engineering activities are not documented, standardized and integrated into organizational work practices of software companies.

This notwithstanding, the local environment represents a significant push factor. There is still a huge unmet and increasing demand for software development services in various sectors of the economy and public services. Despite this and the alarming failure rates mentioned above, compared to the encouraging initiatives in the network infrastructure development and human resource development aspects of the national ICT capacity building programmes, there are no major national or institutional level initiatives worth citing in the area of software development capacity building. Although not commensurate with the huge demands and ambitious projects, an effort is currently underway at EICTDA to develop generic and adaptable guidelines for software acquisition particularly in public enterprises. What is more, as of recent, the need for appropriate and better methods of software production, management and use has been well recognized by all parties concerned and almost everybody is on the lookout for such methods.

1.3 Research Questions and Approach

1.3.1 The Research Questions

Any method will only be appropriate[5] to a certain situation if it is effective and useful to its users, if it coherently addresses the questions of how to organize and conduct an inquiry so that interventions into the problem situations reduce the uncertainties inherent in the problem situation and furthermore enable its users to leverage their experiences. On the other hand, it is not possible to universally prescribe all details (written into processes and procedures) that will deal with the variability of the many influential factors prevalent in any particular development environment (Henry, 1981). Hence it may not be appropriate to expect processes and techniques applicable to all situations, to be defined by methods and process models.

In this connection, there is also consensus among workers that any of the publicly available methods are not design recipes to be applied blindly or literally (Cockburn, 2006**;** Jayaratna, 1994; Mathiassen, 1998; Floyd, 1987). Problem situations in which the designers' works and ways of studying or interpreting the problems vary significantly, and the success or otherwise of the method may be highly contingent. A method may have worked in one situation, but that should not be good enough reason to assume that it will also work in the next. This also applies for different projects conducted within the same organizational context as articulated by Floyd.

[5] According to Korpela (1995), appropriateness is not a pre-determined attribute of a given piece of technology, but depends on usage, affordability, availability, sustainability and needs being satisfactorily met. Pellegrini (1980) maintains that technology should be considered appropriate when its introduction into a community creates a self-reinforcing process internal to the same community, which supports the growth of the local activities and the development of indigenous capabilities as decided by the community itself. As such, for any form of technology to be appropriately developed in a specific situation, the technology is better acquired and adapted rather than simply transferred as is. For our purpose, the interpretation of the term 'technology' is not limited to the design of physical things and artifacts, but also the design of practices and possibilities to be realized through artifacts. It also encompasses the design of new practices (Flores, 1988).

> "Despite all similarities, each software project is definitely different from its predecessors. So for a method to be successful, you cannot simply apply it, you have to work it out in your project"

To qualify in this sense, suitable methods may be selected and appropriately developed to serve in the specific situation at hand.

In the effort to develop appropriate methods for the local environment, while appreciating the differences in the contextual environment and project situations mentioned above, it is also important to note that organizational problems and concerns, regardless of their differences, rarely stand alone and are totally unique. In other words, as much as there are differences, there are many common features, characteristics and tasks that are shared among organizations and projects regardless of their geographical locations (environmental differences). There are also a lot more experiences to be shared and best practices to be adopted with respect to the design and use of development methods. To this end, rather than re-inventing a totally new method from scratch for use in a certain situation (not meant to discourage invention though), an often preferred strategy is to consider selection and then adaptation of complementary aspects of existing methods that are likely to be effective in addressing local project situation. As Gregor and Jones (2003) argued, "Information Systems (IS) as a discipline is concerned with action - the design, construction and use of software and systems involving people, technology, organizations and societies. In the action of building information systems it is preferable not to approach every new development problem afresh."

Selection of a suitable approach for adaptation involves the process of identifying an existing approach (or putting together complementary aspects) from the wide assortment of available approaches – to decide on which methods and processes are suitable for a certain project. This is considered as a very important step in the project design that may make the difference between success and failure. As stated by Kettunen and Laanti (2005:587), "an appropriate process model helps coping with the challenges, and prevents many potential project problems. On the other hand, an unsuitable process choice causes additional problems". In this connection, workers (for instance, Kettunen and Laanti, 2005; Boehm and Turner, 2003; Cockburn, 2006) provide comparative process model selection frameworks for selection from publicly known software process models which project managers can use as a systematic guide for (re)choosing the project's process model. In the local context, there is an urgent need for approaches which presuppose thorough familiarity with local setting and realities including the extraordinary non-standardized work practices in user organizations. The approaches selected must provide for a more rapid system delivery by leveraging emerging developments and best practices in both technological and methodical approaches to maximize late comer's advantages.

Once the selection is made by one means or another, there is a need for a process of adaptation of the selected process to the project situation/context and continuously throughout the development process. In particular, such a process concerns itself with providing decision mechanisms on which aspects of the processes and methods selected be introduced, adjusted or dropped, at which point within the process evolution of a certain project. Such decision mechanisms, among others, should also leverage on continuous basis user/practitioner experiences with the use of the methods and processes in the project. Unlike the selection mechanism, there is generally lack of guideline in this area. In this connection, it is also relevant to note that, even in the area of selection, the guidelines provided are not only anecdotal and sketchy, but also suffer from the context-related limitations discussed earlier. So far, not much has been done in terms of identifying and documenting such contextual factors that characterize and directly affect local software development practices.

From the foregoing, we may observe that the task of appropriately developing methods and processes for local use by tailoring suitable methods and processes developed elsewhere is a challenge for the local software engineering community. This research was, therefore, initiated to explore the possibilities of adapting/adjusting complementary aspects of some of the publicly available software development methods and processes with the dual purpose of enriching the approaches and developing an appropriate methodical framework which is capable of improving the software development situation in Ethiopia. Additional motivation for the research includes the author's interest: to know more about how software process models and methods are selected, adapted and used in practice; and to order and use the author's years of experience in teaching and professional practice in the area of system development for tailoring existing software development approaches to enhance their usability in the specific project settings.

For this purpose, of the multiple questions related to this subject, those identified for investigation in this research work include:

- What does the current software development situation in Ethiopia look like?
- What are the specific software development challenges faced by practitioners in the local setting?
- What are the critical success factors for improving the software development situation (and project success rates) in Ethiopia, and how could these be addressed?
- What are the contextual (national, organizational, technical, etc.) factors that must be addressed by existing software development methods and process models to increase their effective usability?
- What are the limitations of existing software development methods and process models in general and in terms of their applicability to the local setting in particular?
- How can one incorporate valuable lessons learned from years of practical experience (in both teaching and professional practice), as well as the inputs obtained from the efforts to answer the above questions, in enriching and localizing existing models and methods to increase usability of methods, to increase productivity of teams and quality of products?

1.3.2 The Research Approach

Gaining a better understanding of the research questions outlined above, and developing grounded insight on and answers to the questions, required an empirical investigation involving both qualitative and quantitative research methods. While more elaborate descriptions of specific research methods employed are presented in subsequent chapters, the following paragraphs provide a brief and general introduction of the methods employed.

In the quantitative category, in order to assess the current software development situation in Ethiopia, an extensive questionnaire based survey was conducted among local software firms, IT Departments of large organizations that were involved in software development, as well as practitioners (software engineers and project managers) working in these institutions.

Software development is an applied discipline and inherently practical. Therefore, quantitative data from the surveys alone may not provide the in-depth understanding and contextual information essential for revealing how software development projects are practiced in reality. For this reason, in addition to the interviews and discussions conducted to support the questionnaire survey, such practice-related research approaches as case study and action research were also employed to gain more insight. Furthermore, an extensive reflection on the author's years of practical experience both in teaching and professional practice in software development was made. In this connection, in due consideration of the large number and variety of software development projects the author was actively involved in, a selection was made for consideration in this work (see Chapter three for details). The primary sources for information on these project experiences were personal dairies and statements, official communications, reflections with co-workers and the program source codes for these projects.

> In this connection, the following is worth noting. The action research and reflection aspects of the qualitative methods employed in this research followed for the most part the contentions and research practices of Mathiassen (1998) and Heiskanen (1995). As a practitioner who worked very extensively in a number of software development projects for an extended period of time locally, as detailed in Chapter Two, the author can maintain the position of an ethnographer, an action researcher and a software engineering historian in the local setting. This has enabled the author to leverage this position for the purpose of this work.

The surveys, case studies and action research were supported by very extensive reviews of both theoretical and empirical research literature (as can be seen from the References attached at the end of this report), paying particular attention to evolutionary, participative and iterative process models that are based on participatory design principles, and focusing on strengths, limitations as well as challenges.

In addition, attempts were also made to consult available documents and reports pertaining to teaching/learning experiences in the system development area, national policies and programs as well as flagship projects on ICT.

The information obtained from the various sources was organized and analyzed in order to:

- document how software development is actually practiced in Ethiopia,
- characterize the type of software development projects and their context,
- identify the productivity inhibiting factors affecting practitioners in the local setting,
- examine the extent of use of methods as well as the difficulties experienced in their use,
- arrive at what the author argues are the critical success factors in the project setting, and
- establish the need for a context sensitive software development approach to address these critical success factors.

Using this as input and closer examination of the project home grounds as well as suitability to local projects of more recent and innovative methods, an attempt is made to propose a methodical approach that the author claims would help in addressing the critical success factors identified. In particular, the approach proposed is developed by integrating knowledge obtained from practice with the adaptation of complementary aspects of existing methods that were considered suitable to the local setting. Practical experiences from trying aspects of the approach proposed on real-life projects are also reported.

The approach to the solution design could also be said to follow the phases commonly employed in engineering research life cycle but iteratively. Namely,

- Information phase: to collect & characterize information about the current practices, experiences or problems;
- Proposition phase: the initial construction, i.e. modules, theories or prototypes, can be subjected to practitioner and user opinions to provide early feedback;
- Analytical phase: to refine research questions, provide some of the empirical feedback and support the interpretation of the empirical data;
- Technology transfer phase: to package their contribution into a form that is more easily deployed by users; and
- Orientation and exposure.

With regard to testing the applicability of the proposed approach, two lines of exercises were followed. On the one hand, a graduate level teaching supported by real-life project-based practicum (where the author actively participated as an instructor and guide in software engineering, system development and software project management courses) was used as a vehicle for testing some of the ideas and software development approaches that were proposed.

On the other hand, the approach is being used to address practical problems being experienced in the case of the third phase software acquisition project in Organization B. In this project, the researcher is actively working as an action researcher in his capacity as an external expert fully charged with the project design and development supervision. As an action researcher, the author established and worked with requirements engineering group and users to document and develop interpretations of requirements and overall project management.

1.4 Summary of Findings, Results and Contributions

From the experiences so far, inconsistencies have been observed between what outdated traditional methods, legacy technologies and architectures can provide, and the requirements of the ambitious broad-based and transformational development plans under implementation in Ethiopia. These are reflected not only in the delay of project deployment but also in incurring higher overall implementation and operational costs as well as increasing vulnerability to obsolescence and architectural breakdown. According to the findings, unsatisfied users, overwhelming costs, frustrated developers, etc. are still common characterizations of software development projects in Ethiopia.

The general findings also indicate that the use of popular (publicly available) methods and models is weak. Where there are attempts in some corners to use aspects of the popular methods, except in some cases, their use is very limited to the fulfillment of administrative contractual obligations that are still dominated by traditional methods and old practices. Also, a considerable percentage of guidelines used in the course of the development are ad-hoc in-house conventions, mostly not properly documented in written form.

On the other hand, in due consideration of the increasing complexity of the software development projects and alarming rate of project failures currently being experienced in the local settings, although not considered as a panacea for all related problems, there is a general acknowledgement and consensus among both software practitioners and experts in the field that the use of some kind of method to guide the process of development would significantly contribute to improve the existing situation.

The results of this research, in general, are consistent with and confirm the importance of feedback based learning and communication in realizing collaborative development practices. Among the main outcomes of this research and its contributions are the following.

- The local software development situation was documented/characterized, together with critical success factors.
- A project context model was introduced to reorganize the software development life cycle based on major perspectives and undertakings that underlie the principles of participatory design and evolutionary software production. In particular, a complete system development cycle that combines project design, production design and use design aspects is proposed.

- A multi-level collaborative reflective learning model was introduced by tailoring organizational learning and communication models for use in the field of software development projects and continuous process improvement.
- The original STEPS model was further developed to address more software development activities – while the original approach limited itself to providing more of an insight into the software development approach, the revised model which is introduced as 'Reflective Steps' is being developed towards a more comprehensive method and process model. The revised model provided operational level guidelines for activities that were identified in the original model but without detailed treatment. In addition, the revised model included and provided similar details for activities that were either missing or implicitly treated in the original model.
- The approach proposed introduces reflection (particularly, collaborative reflection) at the beginning, in between and at the end of each process step (cycle), starting from project design to production design and to use design, for deliberating on how a particular method is being used and improved in practice.

In addition to its contribution in addressing local problems, this research and the results may also be considered as a methodical contribution to the on-going work on participatory and cooperative design approaches to software development and process improvement. Reflective Steps techniques, particularly the reflective workshops and the supporting learning and communication infrastructure, can be easily used or incorporated in other methods and process models.

The work also raises a number of issues related to capturing and communicating design discussions and decisions. These issues would help to identify and design further research on design communication. Even though more practical experiment is required to concretize, operationalize and enrich the proposed approach, the experience with the method in both teaching and professional practice indicates promising results and optimism that the approach proposed will gradually achieve wider acceptance and significantly improve the usability of methodical approaches that would in turn contribute to the improvement in the productivity of project teams and quality of products.

To achieve this, however, the proposed method needs to be practically tested in a controlled environment. This may be done with the application of appropriate measurement instruments to assess to what extent the methods help in realizing expected benefits. Among the benefits are lower costs, timely implementation, rise in quality, lower defect rates, flexibility to change, and the ability to leverage new technical or business information. Enriched with lessons learned from such real-life project exercises, the effort will also be a timely response for countries such as Ethiopia which are now constantly on the lookout for better ways of developing and managing software projects in line with local needs and priorities.

1.5 Organization of the Report

This thesis is organized into eight chapters. The next two chapters document the software development practices in Ethiopia based on review of selected case studies and a comprehensive survey conducted for this purpose. Chapter Four provides further analysis and discussion on the software development situation in Ethiopia with special emphasis on the identification of contextual factors to be addressed in improving the situation. An experiment conducted in real-life project environment to realize collaborative practices in the local setting is also presented. Based on the findings about the existing situation, and by drawing on complementary aspects of existing approaches and years of practical experiences, Chapter Five introduces the proposed methodical approach for software development and process improvement in the project setting. It also provides theoretical foundations and underpinnings for the proposed approach. In Chapter Six, more details on the application of the proposed approach for software process improvement are presented. Chapter Seven reports on recent experiences from attempts made to apply aspects of the proposed approach in ongoing real-life projects. The conclusions drawn from this research and recommendations for future work are summarized in Chapter Eight. References consulted, survey questionnaire, interview guidelines as well as the various sample source materials cited in the report are attached as annexes at the end of the report.

CHAPTER TWO

2. Software Practice in Ethiopia: Case Stories

Software development is an applied discipline and inherently practical. Accordingly, both quantitative and qualitative data generated from real-life project case studies and stories are believed to provide the in-depth understanding and contextual information essential for revealing how software development projects are practiced in reality. While quantitative data from surveys conducted for this purpose are presented in the next chapter, selected case stories from the author's engagements over the years are provided in this chapter for more insight into the software development situation.

The stories documented in this chapter were particularly meant to help us make good and real sense out of the software development situation prevailing in the local setting. The author strongly believes that such local project stories, when properly documented and shared, may serve as instruments for design and learning in software engineering in the local context. They help us to try out the earlier experiences so we can expand our understanding of the possibilities that are open to us. They help in creating a shared understanding and meaning with others who want to join in the dance of learning and discovering methods and processes that help in the design, development and use software within the local setting. Unfortunately, however, this is not commonly practiced so far. For that matter, even the experiences of the author reported in here were not documented elsewhere in a shareable format. What is more, currently there are a number of application software acquisition or development projects being run by various government agencies almost in parallel and in an uncoordinated manner. Those involved in the implementation of similar projects do not have regular forums where they can share and exchange information and experiences with each other (not even in academic circles). There are no mechanisms whereby such common working documents as system/software design and related specifications, bid specifications and contracts are compared and shared. Project implementation and management works and experiences are rarely published and shared. Lessons (both success and failure) learned in one project (what works, what does not work, what factors contribute more to the success or failure, etc.) are not formally discussed, documented and shared. On the other hand also, the turnover of people involved in the projects is very high thereby extending the learning curve.

This chapter starts by providing a brief information on the experience of the researcher as a background to the cases described in the following sections. This is followed by a description of selected case studies and stories based on practical experience of the researcher as a further source of information in the effort to understand the existing situation. While discussions on factors related to the success or failure of the cases as well as the approach followed in the actual conduct of these cases are presented in this chapter, more elaborate presentations and discussions on distinct methodical approaches employed (to address some of the issues that were not easy for handling by literally applying the methods suggested in the literature) are given in Chapters four and five.

2.1 The Researcher

The academic experiences of the author include: teaching and research at the School of Information Studies for Africa, SISA (currently renamed as Department of Information Science), Addis Ababa University, Ethiopia (since 1990), and at the Department of IT in the College of Telecommunications and Information Technology (since 2005). Among the courses taught as part of these engagements are the following postgraduate level courses: Information Systems Analysis & Design, Software Engineering, Modern Information Storage & Retrieval, Database Systems, and Software Project Management. Courses taught at undergraduate level include: Fundamentals of Programming (first using the FORTRAN, then Pascal and then C languages) and Database Systems. The researcher has an extensive experience stretching over the same period in advising graduate students, particularly in supervising and guiding more than 40 master's level theses in the field of computer science and information systems. The researcher also personally played a leading role in the design of the Information Systems curriculum at the Faculty of Informatics at AAU and the design of the Information Science and Computer Science curricula at the School of Information Science & Technology (SIST) at AAU, and the IT curriculum at the College of Telecommunication and Information Technology.

With regard to professional practice in system and software development, the researcher has been actively working in the area since 1983 in various capacities ranging from a programmer to business process redesigner, software project manager and expert advisor. The programming experience initially involved writing programs to support scientific applications required by researchers and engineering consultants (in the areas of data analysis, simulation and design verification). This required the mastery of the FORTRAN and C programming languages, and mathematical algorithms. Later experiences involved engagement in projects that aimed at developing software for business applications. This required upgrading technical skills in business-oriented programming languages such as COBOL and operating system utilities (with regard to file management, searching and sorting algorithms, etc.). With more exposure to business application development, however, the criticality of such non-technical aspects as knowledge of the business application domain and the importance of working collaboratively with key representatives of users and domain experts were increasingly recognized as part of the software techniques and methods.

Partly because of formal training on methods (as part of the postgraduate studies in Europe), and the advancements in the technology (resulting in the availability of better tools and utilities), and partly because of the need to meet the challenges posed by the huge increase in the number, size and complexity of the application software projects in which the researcher actively participated, as of 1987 attention on the technical front shifted to the use of structured approaches as a means of improving productivity and quality of modeling and programming. The researcher used his university positions to introduce courses in structured methods in the course curriculum. Similarly, the consulting positions were used to introduce these as standard practices in the industry in place of proprietary approaches introduced by computer suppliers. Within the structured methods, more emphasis was given to Data Flow Diagrams (DFD) and Hierarchical Input Process Output (HIPO) charts for process modeling and documentation, Entity-Relationship Diagrams (ERD) for database design, decision tables and pseudo-codes for logic documentation. In parallel to the structured methods, the development and packaging of commonly recurring routines into sharable/reusable forms were widely explored. After a decade of such domination of structured approaches in the local setting, Object-Oriented approaches were gradually introduced as of 1995 in both teaching and practice. While these approaches were used in parallel for some time depending mostly on the skills and preferences of practitioners, over the last couple of years, the dominance of object-oriented approaches is being witnessed.

However, struck by the high failure rates of software development projects in the local setting (including those where the researcher was involved in) on the one hand, and the recognition from experience that the technical approaches alone would not help to improve the situation, on the other, attention was shifted to explore possibilities of developing appropriate methods that would complement and support the efforts on the technical front in terms of improving project success rates. Particularly, as part of the teaching and research efforts, attempts were made to incorporate aspects of project management, user participation, learning and collaboration techniques into methods in the context of real-life project settings. This research in a way is an offshoot of the effort in this direction.

2.2 Stories of Selected Software Development Cases

As shown above, the number of projects that the author was involved in first-hand is high. There is no space here to discuss experiences or share stories on most of these projects. This may not even be necessary. To this end, for the purpose under consideration, the author has limited himself to sharing selected stories of three representative cases that were believed to characterize the local software development situation.

In particular, three cases (of Organization A, Organization B, and Organization C) were selected for analysis and discussion. For ethical reasons, the company identities and ownerships of the cases reported are deliberately kept anonymous. **Organization A** was in the software development service industry. It was a privately-owned small company located in Addis Ababa. It was one of the very few leading local companies in the area of software development between 1991 and 1999. The company custom developed various application softwares for a number of clients in both the government and private sectors. Some of the software projects were reviewed for the purpose of this research. **Organization B** was in the financial service (insurance) industry. It was a privately-owned share company with branches within and outside of Addis Ababa. It had hundreds of employees. It served thousands of customers in all classes of business (life and non-life insurance). Two projects were investigated from this organization. While one of the projects took place between 1997 and 2000, the other project started in 2006 and was an ongoing one at the time of writing this report. The projects aim at automating all the non-life insurance business processes. **Organization C** was in the public utility service industry. It was a state-owned public enterprise with branch offices all over the country. It had thousands of employees. It served millions of customers in four major service areas. The project took place between 2003 and 2006. It aimed at automating the billing and customer care processes.

The selection of these cases has to do partly with the availability of some sort of recorded material in the form of technical reports or documented research (basically theses by students and technical reports) and partly due to the active involvement of the researcher in these cases. Needless to say, the cases also have good materials related to the issues and questions being explored in this research work. Additional sources of information for the cases reported are personal diaries and statements, official communication and reflective recollections of the author.

Throughout the cases, the purpose of exploration was to find out to what extent such global factors as changing requirements, user participation, technological developments, project management and the like did affect the performances of the projects and the quality of the products developed. In addition, in the process of analyzing the cases, attempts were made to emphasize software development experiences that distinguished local practices. The following subsections present the essential extracts from these cases. For better documentation of the learning process involved, the cases were presented in chronological order, starting from the earliest to the latest.

2.2.1 The Case of Organization A

Organization A was conceived and established in 1990 by a group of Ethiopian nationals highly trained in Europe and the US in software and systems sciences, with a view to respond to the fueling demands for software services locally. The company was first established as a consulting company in software system development but then shifted into a software development company by 1993 because of market demands. The author was among the founding members and active participants in the functions of Organization A.

In this subsection, an attempt is made to briefly describe aspects of project experiences at Organization A that involved the development and customization of Integrated Business Application Software (IBAS), a package developed and marketed by Organization A.

> *Important observations: the need to continuously scout out commonality among application modules and develop shared routines/components; the importance of subspecialty and division of labour among the development team (a 'back-office' like subteam taking care of the development of required utilities and reusable components based on the requirements of the 'front-office' subteam, and a 'front-office' like subteam responsible for developing the operational application software based on the requirements from users and using the components and utilities provided by the 'back-office' subteam); the need to continuously scout technology to increase productivity and improve quality.*

A brief background on IBAS was felt in order before the main stories.

In the software development engagements Organization A had with a number of small-scale private companies, two things were observed from early on. On the one hand, most of the requests from users revolved around automating the following business functions: Payroll, Stores Management, Job Order Management, and Accounting. Requests for Personnel Administration and Fixed Asset Management came relatively later in the process. On the other hand, regardless of the clients for whom the software developed and the type of application developed, at the technical level the emergence of recurring patterns and routines was observed. Accordingly, instead of copying and editing these routines from one application to the other or from one project to the other, the developers decided to generalize and develop the routines in shareable formats. Some of the common functional routines identified were: file/data management, transaction processing, report generation, and general utilities to help users secure their installation. On the other hand, because of the high level of interface and data sharing between the above business applications, the developers decided to put them under one loosely coupled but integrated application package and in a manner where the individual application subsystems can also be marketed independently, hence the emergence of IBAS as a product. In the first release of IBAS, packaged among others, were the following applications: Payroll (IBASpay), Personnel (IBASpers), General Ledger (IBASgl), Production Control (IBASprod), Stock Control (IBASstock), Invoicing/Sales (IBASinvoice), Purchasing (IBASpurch), and Fixed Asset (IBASasset). For this reason, although the design, purpose, and function of the applications were comprehensive, each application can be customized to accommodate the unique needs of a particular business firm. This was evident from its successful customization and implementation of IBAS at a number of organizations locally.

With the introduction of IBAS, the technical development group assumed two roles: one of building tools and utilities for IBAS in the functional routines stated above, and that of configuring and developing the business application by making use of the utilities and tools. While most of the former was done at the premises of Organization A, the latter was done at the client site working jointly with users. Moreover, with regard to methods, building on the lessons learned from the success stories in earlier projects (through the experiences shared by the individuals involved in the projects), structured approaches for modeling and programming applications, and collaborative development of the business applications with users at the users' site were employed. Prototyping (particularly, operational prototyping) was employed throughout the projects. In this connection, it is worth noting that the decision to use such prototyping evolved from or was guided by practice (discovered as more practical and appropriate in the process of engagement with the projects) rather than by design based on the recommendations of specific methods.

The database structures, user interface and searching routines for the purpose of building the shareable routines of IBAS were mostly implemented using the C and Assembly languages. The first release of the system was developed to run in the DOS operating system environment as the Windows platform was then not mature and reliable enough. For this reason, to meet the application requirements (particularly to implement better user friendly graphics interfaces), the developers had to use assembly routines to develop a special windows toolkit working directly from BIOS and the graphic adapters. Similarly, drawing lessons from our experience in the mainframe environment and to have 'vendor independent' tools, programmes were developed to implement relational database features by working directly from flat files and using popular file and data structure and search algorithms (which are published in textbooks and technical publications). According to those who participated in the process, these experiences were very challenging but interesting. For historical reasons, sample source programs from these attempts are attached as exhibit in Appendix 2 of this report.

Looking back, the development technologies and environments accessible to the developers locally then had little to offer in terms of meeting the fast growing demands of applications and users. Under the circumstances, therefore, the developers had no choice but to struggle with locally available resources to address the demands as indicated above. However, the situation dramatically changed after a while when relatively easy to use and industry standard programming tools and environments became widely accessible with the popularity of the Windows platform and related application interfaces as well as database technologies and related development tools. Although those who were already involved directly in the development of our original routines and knew the details of the routines by heart found it hard to break the ties, migration to the newer platforms was inevitable, mainly because of pressures from colleagues who found the new tools more robust and friendly. The migration was also unavoidable because of the interest to adopt industry standards which were important to integrate existing products into the user environment (that is, to provide interoperability of our products with other applications running on these platforms that were being introduced to the client environment). Accordingly, organization A decided to migrate to the Windows platform, use C++ and work on the microcomputer-based database platforms available at the user sites (MySQL, FoxPRo, dBase were among those popular in the local market, while Oracle was just emerging).

Selected Case Stories

Important observations from the success stories: *those projects that were successful in terms of delivering usable software systems were typically those where the people responsible, particularly the heads of the departments that own the business processes, worked actively with the developers on a daily basis. Another important observation was the critical role of early versions of operational prototypes particularly in terms of facilitating collaborative work. Users did not value or take seriously use-and-throw prototypes.*

The plan to implement projects in stages (system study and development) was counter productive. Documentation from the front-end system study did not help more than securing a go-ahead with the development decision from project sponsors. The design documentations may have served in deciding on the system architecture and familiarising software engineers with the application domain, but the development team had to re-do this aspect altogether with users as part of the detailed work required to develop the programs. Accordingly, users considered the time spent on interviews and discussion during the system design process as wasted. They preferred the use of operational prototyping as vehicles of requirement definition and design specifications. This, according to the users, gave them the feeling that the software developed was for them as they contributed to the design by identifying and incorporating issues from their day-to-day real-life experience at work.

Taken together, where the users believed that the use of the technology would help them bring improvements in their operation and business performance, they expressed interest and commitment to work with technical people, to provide feedback promptly, to collaboratively develop and implement the required system.

In the process, they gradually developed appreciation of technical issues including the potentials as well as the limitation of the technology. Users contributed a lot in design of: user interfaces, reporting styles and formats, audit trials; and in sequencing of modules (priority setting).

The interest and commitment on the part of software engineers to learn and build their knowledge of the application domain was another success factor.

***Important observations from the failure stories** are the critical role of 'organizational communication' and 'boundary management', and limitations or lack of competence on the part of the development team in these areas. Throughout the projects, because of good working relationships and understanding with user counterparts (particularly project sponsors), the development team took the user support for granted, soon to find out that this was only the case as long as the initiators of the project from the users' side were active and had the required authority until the end of the project. No attempt was made to create necessary awareness among other members of the user organization on the project details and to solicit their support for the project. In most of the cases, communications from the development team were restricted to the user counterparts assigned to work on the project. For the most part, the development team was more concerned with technical issues and confined to manage things within the project team. No attempt was made to address issues in the project environment. In effect, very few people knew about the projects. In almost all of these cases, the local situation was that the project existed as long as the initiators existed; where the initiators left before the completion of the project, the project ceased to exist.*

What is more, the causes for some of the failed cases related to lack of proper definition of scope and priorities. Attempts were made to address whatever the users expressed in one go, without properly scoping and prioritizing with provisions for learning from one application to the other. As indicated above, the separation of system design from the development and the related rigid contract *based on strict procurement and budgetary constraints had also contributed to the failure.*

Among the examples of projects that were considered successful from the business engagements made through organization A were: a cost accounting application software development project for a printing press, the customization of IBAS to a metal and wood furniture manufacturing enterprise, a staff loan and credit management system (a sort of mini-bank multi-currency application system) development project for an organization in the United Nations system, and an insurance accounting system for Organization B. In each of these cases, the mode of collaborative work arrangement that evolved in the process consisted of a development subteam composed of two software engineers working with one domain expert from the user organization (a somewhat extended version of what is now referred to as pair programming in the agile movement). In most of the cases, the author took the role of one of the software engineers.

In all these cases, the developers were given desks at the user site and more or less developed the application system on the users' site using operational prototypes. The developers spent the normal working hours with the users every other day and used the days in between to work with the back office technical team to prepare necessary utilities to implement the functional requirements and/or non-functional features as agreed with the business process owners. The new features or modifications incorporated into the operational prototype were then tested with the users the next day. To expedite the process, it was agreed to use common test cases from real-life business transactions, where user counterparts manually workout details of the test cases and software engineers implemented related routines in the prototypes to facilitate cross-checking process understandings. For instance, where calculations on interest rates, discounts, taxes, etc. are involved, users take samples and prepare a worksheet where the details of the required computations are worked out manually with the intermediate results clearly written at each step; where requirements relate to reports, users prepare sample formatted reports with inputs and outputs clearly worked out manually on paper, etc. As indicated, to support this mode of development, operational prototypes were used. And to facilitate cross-checking during the joint sessions with users, the prototypes included required tracings to display intermediate results. In this way, the software engineers and business process owners jointly walked through the revised version and confirmed the implementation after making required modifications, omissions and inclusions. Where incorporation of additional modules or modification of existing ones required more technical work, developers took some days off to work with the back-office team and come back to users to test the revised versions.

In other words, there were two development sites and teams, one was at the premises of Organization A developing required utilities and tools, the other at the user site developing the business application (user interfaces, business logic, storage and retrieval of data, etc.) with users. In addition to expediting the process and creating mutual understanding between software engineers and the domain experts, this arrangement of work helped both parties to build knowledge about each others' domain and business processes. In the process, users were not only able to easily articulate additional requirements based on practical knowledge of what is possible and what is not, but were also able to contribute to the design of artefacts required to increase the quality of the systems and in some of the cases to help in the modification of certain features and functionalities (mostly in the area of generation of user-defined reports, and routine maintenance activities) using utilities provided for these purposes.

Among the examples of projects that were not successful from the business engagements made through organization A were: a project initiated to design organization-wide document management and office business automation for one of the ministries of the Federal Government of Ethiopia, a hotel management software for one of the national hotel chains, and the customization of IBAS to an electronic retail and distribution enterprise. While the first two organizations were state owned, the third one was privately owned. Due to the requirements from the user organizations, which had more to do with management control related to procurements processing, in the first two cases the projects were planned for implementation at two stages. The first stage work involved studying the existing system and redesigning the business processes to help overcome the limitations of the existing system. The second stage concerned the construction of the software required to support the redesigned processes. In both of the cases, while the system design works were successfully completed, the software development aspects were unsuccessful. For one thing, in both cases, it was not possible to immediately embark on the software development activity as the approval of same took months. This was partly the case, since the external advisors employed by the client organizations to supervise project activities took more time to give their feedback. By the time the go ahead decision was reached, some of the technical people involved in the study were assigned to other projects or left the organization. Accordingly, more or less a new development team was assigned to work on the development project. As expected, the new team required sometime to study the design documents and familiarize itself with the application environment.

In the mean time, similar changes also took place on the client side. In the first case (at the ministry), the development work became a protracted exercise particularly when the user champions that initiated the project and that were keen to overhaul the processes were transferred to other units within the organization and away from the head office where the project was stationed. There was also major organization-wide restructuring and reshuffle of staff which made it difficult not only to keep the momentum of the project progress but also getting timely feedback from users. Because of additional workloads on the replacement staff and lack of required support from the in-house IT departments (who in fact did not demonstrate enthusiasm from the outset for the software development initiatives with such outsourcing arrangement), and the interest to cut down related expenses (charged by the developers because of the schedule extension), the development project was cancelled after going through a couple of test iterations.

With regard to the second case (at the hotel), there was not much change that took place in the client organization. After spending some time to study the design documents, the development team was able to come up with an initial prototype for review with users. However, it was not possible to get hold of domain experts to comment on and enrich the prototype. Among the reasons given were, that they had much operational work to attend to and under the circumstances could not afford the time to work on the software; that they had already communicated their requirements during the system study and thus did not have any additional requirements; and above all they doubted the realization of the automation plan because of the delays experienced to procure the required hardware because of the failure of the hardware vendor to come out successfully from the assessments by the external expert group. According to users, particularly the last factor was good enough reason to authorize the transfer and use of the annual budget allocated to the project (before it expires) to some other lines that already used allocated budget effectively and were asking for more. On top of this, the rumour that the General Manger of the hotel, who initiated and supported the automation project, was to be transferred to another enterprise also contributed to lack of enthusiasm among users. As a result, after six months of delay, the client management decided to postpone the automation project for some other time.

Still closer examination of the two cases revealed that these projects were larger in size and complexity as compared to those in the success stories described above. Each project involved a number of full-fledged packages in two broad categories: business functions (such as human resource, payroll, accounts, asset, etc.) and professional services. There were a number of functional departments involved in the process. Each project had branch offices scattered all over to be considered in the requirements analysis. Unlike the privately owned client organizations, these government owned client organizations had strict and rigid procurement and budgetary constraints to operate under. In this connection, lack of appropriate project management methodology to address issues related to project scoping, prioritizing applications to be developed and coordinated among parties involved in the project had also partly contributed to the failures. The developers had no relevant experience in these areas to help users. They also did not at all consider it appropriate to take part or responsibility in these areas as part of their main activities (these were totally left for users to handle).

The third case in this category, the project with the retail enterprise, was initiated by the Accounts Manger of the enterprise, who came to Organization A based on the recommendations of other clients who had successful engagements with Organization A. As such the automation project for this enterprise was jointly developed and a contract was signed after negotiation. The development project basically involved customization of IBAS to the client environment. After going through a number of iterations of customization based on the specific requirements of the client, the first operational version was installed for trial at the Head Office. At this point, a new request came to develop a Video Rental System (VRS) with high priority as this newly introduced business met with difficulties with the manual means (particularly, the number of customers and transactions increased at alarming speed). The VRS was successfully developed within three months partly because users were desperate for such a system. The deployment and use of an initial version not only helped increase efficiency but also boosted company image among end users who were very impressed with the speed of processing and level of detailed information provided to users on the spot at checkout.

When returning back to the IBAS customization, after four months, the developers were required to almost go through the entire customization all over again. This was partly the case because most of the user counterparts that worked with the developers had already left the company or transferred to other units and their replacements were in fact new college graduates who were keen but not very well familiar with the business processes and the versions of the software deployed earlier. What is more, there were also groups among the users with vested interests in the status quo - that favoured working with an existing application software that was to be replaced by this project. The four-month absence gave this group an opportunity to reinstate the old system on grounds that the new system was not yet tested and ready for them to close accounts and produce reports for management. This resulted in a serious negotiation between Organization A and the client organization as the effort to put the project back on track required additional time and cost. In between this, the key negotiator and project initiator (the account manager) got seriously ill. The negotiation was interrupted as the client organization insisted that finalization of the negotiation to reinstate the project be pending his recovery as he was the one behind the project from the very beginning and who was assigned to act on behalf of management on matters related to the project. Unfortunately, the account manager never made it back – he passed away. This, in effect, also resulted first in the postponement and then cancellation of the development project.

> *With regard to methods, organization A adopted the use of such structured techniques such as data flow diagrams for process documentation, entity-*

relationship diagrams for database, decision tables and pseudo-code for logic description. For information gathering, interviews with selected user representatives and documentation reviews were used. In practice, however, the data flow diagrams and entity-relationship diagrams only served the purpose of orienting the development team with the application domain and designing initial versions of the system architecture. In some of the cases, where there were contractual requirements, it also helped in the preparation of system reports for management consumption as an evidence of progress. The decision tables and pseudo-code helped the technical people (software developers) to get more insight into the details of the business domain (in other words, for the purpose of training the technical people) to enable them produce an initial version of the prototype. What is relevant to note in this connection is the fact that, throughout the various projects, although the initial understanding was to work out the business requirements as detailed as possible first and then develop an operational (or close to operational) version of the software for testing, this had never been the case.

The developers learned the hard way that this was only possible by working jointly with users from the very beginning using prototypes. The actual processes can only be discovered from transaction stories and test cases completely worked out with users. Particularly, the work out examples which are done by users manually and by developers through software components in the prototype and the step-by-step walkthrough supported by tracing were very useful instruments. While this approach served the development very well, the difficulty experienced in this connection related to the documentation of the design processes and the results, particularly the intermediate results and alternative design options considered before finally agreeing on the implemented version. According to the practice at organization A, while comments in the source code were used for the final prototype versions, version controls were used to capture intermediate design options. With regard to more elaborate design options, the details were captured manually by simply jotting down notes on the worksheets often in *unstructured formats - mostly influenced by the styles and handwritings of the particular experts. (see Appendix 3 for samples). As the size of the routines increased, easy reference to and retrieval from both the electronic and manual records became more difficult. This problem persists to date.*

2.2.2 The Case of Organization B

There were three projects with organization B. The first project related to custom-developing an insurance accounting software. The second project related to custom-developing comprehensive insurance application software based on the results obtained in the first project. The third project related to the customization of off-the-shelf insurance software. The third project is still an on-going one at the time of writing this report. It is also one of the projects where the newly proposed approach is being tested. Thus discussions related to the third project are presented in Chapter Seven. The first project is similar to those discussed in Organization A above. The discussion in this subsection, therefore, relates to the second project.

The main observations drawn from this story are the advantages and disadvantages of engaging and depending on highly qualified and experienced developers and domain experts; dangers of accommodating changing user requirements continuously throughout the project; and the risk of technology adoption without proper assessment of consequences. Obviously, engaging highly qualified and experienced experts in the development of software may have many advantages. Those that stand out in this case relate to: faster development of quality modules, flexibility in accommodating requirements, better understanding among and within the development groups, and most of all detailed documentation of the business process redesign which served purposes beyond the development of the application. For instance, the documentation produced as part of this process has been in use for long for practicum in teaching software development. On the other hand, the feasibility of conducting full-fledged business redesign work as part of the software development activity; the possibility of a domain expert turning into a software designer or developer; the *possibility of a software expert gradually assuming the role of a domain expert were important observation that may be extracted out of the project experience with organization B.*

On the down side: the scale of change was enormous and the emphasis on precision and perfection was high, the excitement to exploit new developments in the technology and the enthusiasm to come up with an international standard and best in class application package were all high, to the extent that none of the versions developed as part of this project made it to real-life operational environment. The development took the shape of a research work done to satisfy the high expectations of the developers and domain experts rather than delivering a product to support the business processes of the client. The other major drawback was extreme confidence and dependence on few individuals, which resulted in putting the project in danger when some of the key experts left the project for reasons beyond their control.

As indicated above, this project was initiated due to the successful completion and use of the first project that concerned the development of an insurance accounting software. This second project was relatively large in size and complexity. The plan was to develop a software application package that caters for all classes of businesses (life and non-life) and to automate every aspect of the business process ranging from proposal processing, premium calculation, policy issuance, policy maintenance, claims processing, coinsurance and reinsurance management, to accounts management.

The second project drew lessons from the first project particularly in terms of the work involved in programming and related technical feasibility. The domain expert assigned to work on the project, in consultation with his professional colleagues in the insurance business and the management of Organization B, took the initiative to re-design the business processes as part of the effort to develop full-fledged industry-standard insurance application software. This task required the study of similar processes in the other local companies and best practices elsewhere in the industry. It also involved exploring the possibilities of benefiting from new developments in the technology to implement better functionality and interfacing.

In this project, the domain expert was not only acting as the business process expert, but was also given mandate to interface with the development team on behalf of the users and also played the role of project leader. As a highly qualified and experienced person in the industry and a person keen to see the development of home-grown quality software of international standard, the expert devoted himself fulltime on the software development project. The author had the role of a chief technical designer and counterpart to the domain expert from the software side. Together we worked for almost two and half years on the project. Based on his experience in the first project and extensive almost day-to-day meeting with the author (and with the software team to give feedback on the prototype), the domain expert became fully conversant with software design methods. In fact, through the process, the role of the domain expert shifted to a developer and decision maker on software design options. To incorporate best industry practices, the development team was assisted by another team from outside Ethiopia which was led by a highly qualified and experienced insurance business man, very active at the time in the African Insurance Association. This external team not only contributed to the design of international standard software because of international exposure and the rich experiences of its members in the industry, but also developed an interest in the marketing of the product at a regional level in Africa.

Working with such arrangements, impressive results were obtained at the end of the first eight months, particularly in terms of documenting the redesigned business processes on the one hand and building the knowledge of software experts in the insurance business area on the other. As part of this process, insurance processes in the various classes of business and functional units were documented in detail together with the decisions on the design options. The knowledge base was captured for the most part on paper in simple natural language text and diagrams/charts for anybody to read and understand. (Samples of such records are attached as exhibit in Appendix 3). By then, a revised version of the existing insurance accounting software was released after correcting bugs in the first release and addressing some additional requirements from the operation group at Organization B. In parallel, migration from the DOS environment to Windows, from a PC-based to Server-based platforms and from C to C++ routines were also planned.

The preparation for this migration, particularly familiarizing oneself with the new technical development environment was not an easy ride. The technical people, while studying the insurance business domain on the one hand, were also extensively engaged in training themselves with the use and application of the new programming tools and techniques. The growing size of the source code in the C version (which was close to one million steps for the applications implemented and the various library routines) and the incompatibility between the old and the new platform made the migration more complicated. Most low level routines had to be completely redone using the features of the new platform. Business transaction processing related routines had to be carefully transferred and tested. At times, with the increased enthusiasm and resulting pressure from the sponsors, the task became very frustrating. To reinforce and expedite the process, another technical team was organized that involved more talented and skilled C++ programmers. Members of the new development team were not only former students of this researcher, but had worked with the researcher on other earlier software development projects. Although their mastery of the new platform and programming skills were very good, it took more time to familiarize them with the insurance domain and to bring them up to speed in the project. Nevertheless, with the reinforcement, towards the end of the first year, we were able to demonstrate early versions of the prototype for some of the modules in the new platform.

The impressive documentation of the redesigned process and the demonstration of an early version of the prototype, of course with better features and capabilities of the new development platforms, increased hope and expectation on the side of our business partners. What is more, through contacts made by the external group, the project attracted more regional/African attention. With this development came request for another major revision of requirements. Among the major requirements was the need to revise the system architecture in such a way that the general insurance business processes common among insurance companies be pulled together as the core component of the base system with adequate provision for customization to each company preferences. This regional interest in the project culminated into an agreement to upgrade the system to meet the revised requirements. In particular, this agreement is to upgrade the features and capabilities of the software to a full-fledged off-the-shelf package ready for deployment in small to medium insurance companies all over Africa. According to plan, organization B agreed to serve as a test bed and the regional association to finance the development of the software. To this end, the redesigned process and the prototype under development had to be revisited all over again.

To cut the long story short, the migration and upgrading project went well up until the sudden departure (away from the country) of the domain expert (for reasons beyond his control) towards the end of the second year. By then, the current researcher was preparing to go abroad (to USA) on a sabbatical leave after making necessary arrangements to transfer core responsibilities to another senior software engineer. Per the arrangements, the researcher would continue to work on the project through the Internet.

After the departure of the domain expert, the work arrangements for the insurance software development had to change. Before the departure of the domain expert, the developers and domain expert were in day-to-day and face-to-face contact, sometimes sitting together in front of a development workstation to test routines and make on-the-spot corrections and modifications. Any business concept that the developers wanted clarification on were explained with practical examples from the real-life experience right on the spot using worksheet as a tool by drawing on similar experiences from Organization A. Access to required documents and business specialists were readily available on site. Developers were also motivated and encouraged from the challenges posed by the domain expert, prompt feedback on any request, the assistance in the testing of converted modules and visits by prospective buyers of the software from the local and African insurance company representatives (who often expressed eagerness and support to use a product designed by ‘Africans’ for ‘Africans’). This situation changed after the domain expert and project manager left. Although attempts were made to maintain communication and follow up support from remote, it was not easy and convenient.

Based on a series of consultative meetings held among the concerned parties (the development team, the external group, the management of Organization B), a decision was made for this researcher to take over the project coordination and liaising with other domain experts from Organization B. The researcher was not new to the industry as he had an opportunity to work for the Ethiopian Insurance Corporation (the state-owned and largest insurance company in Ethiopia) as the Head of the EDP Department (as it was called then, now changed to IT Department) and also worked very closely with domain experts for a couple of years as a software engineer. In this capacity, working from USA, the researcher tried to support the technical team through email and telephone. The emails, exchanged between the researcher and the development team, were very extensive and technical. Although the documentation produced after the redesign were detailed enough to explain most of the issues raised by the developers, the developers on the local side required additional explanations from the researcher basically for two reasons. Firstly, they could not afford the time (off the development work) to read the documentation. Secondly, in the course of implementing the design they usually came up with alternative designs that they felt were better but wanted assurance on whether these would capture/support the original design considerations. As such, the emails involved answers to various queries from the developers, additional clarifications on the design specifications prepared by the domain expert, additional requirements, and mostly design options for the realization of software modules which were not straight forward to implement or for which the developers have invented better ways than originally proposed. Together with the detailed and very well articulated stories and design specifications prepared by the domain expert, copies of these extensive email texts and attachments served to this date as a reference to the application area in both teaching and practice. Sample copies of the design specifications prepared by the domain experts and the email exchanges are attached in Appendix 3. It was not possible to attach the complete copy because it is a big volume on its own.

After a year of work in this manner, the project took yet another shift in the development direction. By then the project enjoyed high marketing. But it was difficult to get the first version to the market. In the mean time, after making an assessment of the progress on the one hand and the market demands and time to market on the other, the sponsors decided to make another major revision at technical level. In particular, a decision was made to develop a full-fledged web-based version using Java. To expedite the process, to support the local development team, and to better market the product, a decision was also made to establish an off-shore software development company with a merger arrangement with a development team from the Middle East already working on a similar project. The development work continued along this line for some time. However, the author was unenthusiastic about the terms and conditions of such deal, particularly the commercialization strategy that in a way demanded complete detachment from a university environment and hence the research exercise on development methods with students. According to the information from colleagues that have joined the effort, the development project continued by involving more software engineers and insurance experts from both Ethiopia and the Middle East. Still the efforts of the project team to release an operational version to the market per plan met with difficulties.

After waiting for sometime, however, the management of Organization B decided to no more wait - as it could not afford to continuously wait amidst the growing competitive environment and increased business volume and pressure from partners and stakeholders. Accordingly, it discontinued its relationship with the project and decided to look for another means to meet its demands. This time around, Organization B opted for exploring an alternative strategy to acquire the required software – in particular to procure a commercially available software package. Because of the reputation established already during the earlier attempts, the familiarity of the case history and technical support provided during the difficult times to maintain the previously developed software, the management of Organization B called upon the author to advise and help in the realization of the new strategy. Such is the beginning of the third software project at Organization B. This time, in consultation with and upon the approval of the manager, the author tried to handle the assignment by applying the newly proposed approach in this research. The experience from this effort is reported in Chapter Seven of this report. The lesson learnt from this case study have already been summarized at the beginning of this subsection.

2.2.3 The Case of Organization C

There are two stories, worth sharing from Organization C. The first relates to sharing the experience from a project initiated for the acquisition, customization and implementation of an off-the-shelf software (where the author first took the role of the project sponsor, then as troubleshooter, and then took the role of an investigator (together with his graduate students) after leaving Organization C. The second case relates to sharing the experience from a business process re-design project that was initiated as part of a corporate reform program and as a front-end for the automation plan that involved application software development. This subsection deals with the first case. The second case is presented in Chapter Four in due consideration of the contribution this experience made to the methodical approach proposed in this research work.

> *The main observations drawn from this case are: the importance of a participatory approach in the sense of Dahms & Faust-Ramos (2002) – and the drawbacks of instrumental participation in software development; the importance of organizational competence among project team members as a critical success factor in software development; the serious negative consequences of lack of organizational communication on project related matters. What is more, the fact that large-scale introduction of software systems in organizations result: in changes that commonly lead to new ways of working and shifts in social relationships in the workplace, in altering the ownership of and patterns of access to information, in affecting established protocols of decision-making and the exercise of influence by individuals and groups; and as such the need to introduce large-scale systems step by step and accompanied/guided by appropriate change management and organizational embedding techniques.*

This project was one of the main organization-wide reform projects initiated to enhance service efficiency and revenue maximization. It particularly aimed at replacing the legacy customer service application software. The system to be replaced was a centralized system, developed in-house using an RPG (Report Program Generator) language. The system used to run on an IBM AS/400 midrange computer platform. When the existing system failed even to handle the billing service properly for technical and other reasons (Ahmed, 2006; Getahun, 2006), and when it failed to cope up with the increase in size and type of business as well as customer service quality expectations, a decision was made to replace it with a modern and robust system that addresses not only the deficiencies of the existing system but also takes advantage of the advancements in the technology and best practices in the utility industry.

To that end, a technical requirements specification document was prepared for the procurement of an integrated and convergent solution (both hardware and software) that was flexible and scaleable enough to support the existing and forthcoming services provided by the utility company. According to the bid document prepared based on the requirement specification, the project was expected to cover a range of activities, including: examining and studying the nature of the existing system, analysis of future requirements, the supply of both hardware and software solution to meet the requirements, installation, testing as well as commissioning of the systems, data conversion and post implementation support. The tender was floated as an open, international competitive tender. The first two tendering processes were not successful: the first because of inflated offers well beyond the allocated budget and the second because of absence of eligible vendors to meet the overly specialized revised tender requirements. Accordingly, the tender was revised again and floated for the third time. About fifteen suppliers responded to this third tender. Four bidders came out successfully from the first level screening based on the preliminary evaluation criteria published in the tender document. The offers of these vendors were further subjected to technical and commercial evaluation, in accordance with the evaluation process published in the tender document. As a result, a company that claimed to be a global provider of the required solution, won the tender and went into agreement after a thorough negotiation on both technical and commercial issues.

Per the contract terms, the supplier conducted a site study process together with key representatives of Organization C which resulted in a detailed statement of work. Based on the statement of work, an appropriate project organization was jointly developed that required both parties to assign senior personnel each to act as project manager on behalf of its respective organization. Other elements of the project structure were also staffed accordingly.

The project was originally planned for implementation within nine months. But in actual fact, it took well over eighteen months to operationalize some aspects of the software (Ahmed, 2006; Getahun, 2006). In addition to the delay, obviously there were considerable cost overruns. Serious difficulties were also experienced in implementing certain modules. Users had serious complaints on the software features and capabilities. Taken together, both the implementation and use of the software were very problematic. Benefits expected at the outset were not realized to a satisfactory level. In effect, the project became a very controversial issue throughout Organization C and its clients. In the following paragraphs, an attempt is made to highlight some of the problems encountered in the implementation and use of this software with a view of drawing findings relevant to address the questions raised in this research. Further details of the project, including detailed analysis of the problems encountered and possible causes of these problems are documented in Ahmed (2006) and Getahun (2006). The studies cited were conducted as part of a master's thesis under the supervision of the author of this report. Beyond citing some of the findings from previous studies, an attempt is made below to further analyze selected issues in more depth than ventured by the other studies.

One major conclusion to be drawn from the information gathered as part of the above cited studies conducted by the students, is that users did not feel consulted about their work needs in the course of the software development work. From the researcher's knowledge, such complaints were made even when participation by those functional units responsible for the business application supposedly happened, as explained above in the project organization. Closer analysis indicated that this user feelings may have come from the fact that key user representatives (those that were responsible for the functions and who did actually participate during the design) did not make it to the implementation stage. According to the inquiry made to clarify this, it was found out that most of the earlier representatives that participated in the process were either removed from their positions (as part of the management reform program that was underway at about the same time when the software was being developed) or had already left the organization for better opportunities elsewhere. In this connection, it is also relevant to note the following reality being currently experienced in most organizations (including Organization C) that are undergoing reform in Ethiopia. Newcomers that assumed offices through the management reform programs were less inclined (or did not feel comfortable right away) to take up and pursue project initiated by their predecessors. The usual practice or tendency was to revisit every initiative and where possible to discard the ongoing project and initiate own versions of the project. The project under consideration was not an exception. Other viewpoints on user participation are provided below.

When we look at the background of the project initiation, as described earlier, there was not only a legitimate reason but also an imperative need to replace the old system. The old system was developed in-house but could not any more cope with the increasing expansion of existing services and the introduction of new services. Review of management meeting minutes and consultation on the subject indicates that the maintenance costs were also high; the technical support was almost nonexistent; there were lots of complaints from both customers and operators on the delay and errors in the bill processing; etc. In short, there was an incontestable need for replacement. What is more, as indicated earlier, repeated attempts were made to attract internationally reputable and best in class companies for the supply and customization of a commercially available system but all in vain. It was only in the third round of such tender that it was possible to get a supplier who came out successfully from the bid analysis process. The tendering and contract negotiation process took quite a long time. What is more, the customization time proposed was also long, due mainly to the size and complexity of the required system. During this period, while maintaining the legacy system to handle the very essential aspects of the billing process, an in-house group of young software engineers were assigned to later develop software system that took care of (rather rescued) some aspects of the required system. Members of this development group and key representatives of the service units were in fact the ones later assigned to work as counterparts to the external developers in the project.

In view of the foregoing, perhaps another factor for the users' feeling of not being consulted that could possibly be inferred from this project setting, emanates from the closer relationships that developed between the external developers and the local counterparts that represented users over the course of the project period. Due to closeness to the external developers and detachment from the users, users must have felt that the staff assigned to work as counterparts to the external staff no more represented their interests as they spent too much time with the developers and did not engage users as they used to do, particularly as compared to the experiences during the course of developing the in-house system mentioned above. Through time, it was observed that user project managers and in-house IT experts that worked as counterparts to the external group developed some sort of arrogance to the extent of acting as barriers in the establishment and maintenance of direct communication between the developers and the ultimate end-users. Towards the end, for instance, there were unequivocal manifestations from the in-house experts working as counterparts to the developers that direct user involvement would not add any more so far as the requirements were concerned. This was made on the grounds that the counterparts themselves have better knowledge of the requirements because of their long involvements in both the previous (in-house) and current projects and long years of service experience with the workings of the organization. The experts also resented any move and intervention by external or independent consultants to clarify such misunderstandings or to make them change their mind on the importance of active participation by users for the better of the project.

The other possible observation that may also be drawn from the data collected on the project experience relates to the difficulties experienced by users in the effective use and operation of the features and capabilities of the newly developed system, particularly those features and functionalities that were nonexistent in the legacy system. The new software deployed included new features and capabilities that were included to support newly introduced businesses and best service practices in the industry. This difficulty in the use eventually resulted in serious complaints and resistance to own the system which in some of the cases culminated into demanding the reinstatement of the old system as an interim solution up until users familiarize themselves with the features and functionalities of the new systems and feel comfortable with its operation and use. Most of this can rightly be attributed to the lack of proper and timely design and implementation, and of awareness and training programs for the newly introduced systems. Although training was provided, according to users, it was inadequate both in content and intensity and it came relatively late after cutover. It was observed that when appropriate training was given to users as corrective measures, the number of enthusiastic users increased dramatically. This notwithstanding, of course, some of the resistance was also to be expected when introducing a system of this scale in an organization that did not see any major change in its existence for long. Allowances should have also been made for users as different categories of users differ in their ability to learn and adapt to new systems. All these use design related issues should not have been ignored and should have deserved explicit treatment during the planning and implementation.

Perhaps at the root of the causes for the problems around the project is failure to anticipate (reasonably predict) the effects of the introduction of such a comprehensive and core (to the business) system on the organization structure, the jobs and roles of the workers directly affected by the system. The project placed much emphasis on the technical aspects of the system. Both the management and the developers (in-house experts included) failed to recognize that the users who were to be directly affected by the system might not see systems in such a benign way as they did themselves. In reality, what was later discovered/learned the hard way was that most of the users, particularly the majority of the labour force, perceived the introduction of the system as instruments of management repression. Some considered the new system would in the end deskill their jobs. In this connection, it is relevant to note that the software was being introduced at the time when many corporate wide organizational change initiatives were underway including the overhauling of the infrastructure at breath-taking speeds and reengineering of the business processes, as well as a result-based employee performance management system (all of which were not taken positively by the labour force).

One must not also ignore the fact that the impacts of introducing a system of this scale in a huge organization as Organization C in one go are widespread and yet often subtle. For example, as is common in most developing countries (Dada, 2006; Heeks, 2003, 2002), the type of easy access to information provided by the newly introduced systems was not welcome to the bureaucracy. With the introduction of the system, it was possible to know on the spot where there are ideal resources to be made available for customers; it was possible to track the performances of sales people on a daily basis; it was possible to follow up fault conditions and the performances of the maintenance crew; it was possible to log customer complaints online for immediate review of same by both customers and supervisors; it was possible to provide not only one-stop shopping service to customers but also 'no-stop' shopping service where able customers (with internet access) would handle their business dealings with Organization C directly from their workstation; etc. In a bureaucracy typically characterized by a limited degree of openness and trust, limited delegation of authority, more interference with work activities, etc., the introduction of such system was no wonder perceived as a threat with a potential to erode the power base of those who have formerly had control over its flow. From this perspective, the introduction of the system became more of a political intervention that required an appropriate change management program as part of the system implementation plan, well beyond the technical-oriented project management that was in place.

As indicated earlier, it is also relevant to note that in the project period (counting from the time management decided that the existing system be replaced), 2002 to 2006, Organization C had gone through two major structural changes. With the organizational reform plan demanded by the owner, a third restructuring was underway at the time of reviewing the case being reported. This trend is expected to continue as the organization preparing itself for the upcoming competitive environment both at national and regional (African) level. Accompanying such restructuring was also the series of management reforms that resulted in the continuous reshuffling of managers and key personalities. In the process of such management reform, those who initiated the reform projects in general and the software project in particular left or were removed from their positions before the completion of their respective projects. As such, developing systems under such conditions was often difficult and at times problematic unless appropriate approaches were devised and continuously adapted to the changing environment usually based on experience and precedents.

The process model adopted for the project was more or less waterfall. Following the signing of agreements, the representatives of both companies met to work out the scope of the work in consultation with the concerned functional units. Required information for customization was basically collected using forms specially designed for this purpose. The forms together with the information collected were then forwarded to vendors. Using this information the vendors tried to customize the base modules of the software. The customization work took place (away from the user site) in the development site outside of Ethiopia. This was followed by further modification and testing of the customized version at the client site working with user counterparts. In parallel with the modification and testing, installation of modules and data conversion processes were initiated. This was followed by user training and commissioning.

Throughout, the understanding on the vendor side was for in-house business processes to be changed to fit into the software requirements as the software was expected to introduce best practices to service delivery. On the other hand, users preferred to stick to their in-house practices and as such expected the software to be customized to support these practices. In fact, at the time the software was being introduced, users were already in the process of revising their business processes as part of the corporate-wide reform program. As such, users did not see the need for (at least not prepared to make) the sort of further changes demanded to operationalize the new software, particularly when this was brought to their attention at the end as part of the operator training.

Analysis of the situation reveals two conflicting issues. On the one hand, the use context was considered important and thus the software should cater for the specific requirements of the users. On the other hand, in line with the reform initiatives, particularly the need to transform the service provision by introducing best practices, if the features being introduced by the software were better, the existing processes should be redesigned to fit the software requirements. All these details, it was discovered, were not looked into adequately at the time of customization. That is, the decision whether or not the application software system to be customized should fit into the existing business processes and work practices was not at all addressed. Moreover, whether or not the business processes and work practices should change to fit the customized software which was expected to introduce best practices was not addressed in timely manner. Despite their claim of international exposure and rich experience in the customization and support of the software to many environments, the vendors seemed to have failed to see the software within the operational and use context where the customization could not be made in isolation from the context.

User participation in the project, touched upon earlier from another angle, was minimal. Users did not get the chance to directly participate in the customization – this was done indirectly through supply of their requirements on the forms provided and indirect consultation through the project counterparts. Users saw the software developed for the first time during the training conducted to enable them to operate and use the system. Most of the people who directly participated in the customization process were technical project counterparts from the IT support unit of Organization C. Although these people had a good deal of operational knowledge in the business activities, mainly due to their involvement in similar earlier automation projects, there were a number of customization issues that were not considered important by the counterparts but really mattered for end users. For instance, users complained about the level of details to be captured on screen forms while registering a customer in the new system. This according to them was not only in a marked contrast to the previous practice in terms of time required to enter the details, but also included unnecessary details, some of which could easily be handled during back office processing at a later stage. The lengthy registration process also resulted in lots of complaints from customers.

In general, the developers used traditional methods that gave users essentially a passive role, limited to serving as informants. There was no meaningful direct interaction and participation by users beyond filling out forms and attending project review meetings which were mostly dominated by complaints from the suppliers that deadlines for sending documents were not met. There was no direct engagement with the front office staff who were to use the system in their dealings with customers.

There was too much rigidity on both sides with regard to following contractual terms quite literally even where there was a need for flexibility (there were discussions right from the date of completing the scope of work on the incorporation of additional requirements discovered in the process and settlement of claims for compensation resulting from the additional requirements). There was lack of coordination and information sharing among the application software vendors and platform providers (there was continuous unsettled argument between the main software supplier and another subcontractor to support a software module). The functionality as well as maturity of certain modules was not tested at all (some core modules were to be newly developed for this project). The situation was further aggravated with serious conflicts that cropped up between the in-house technical team (who often tried to show authority) and the technical team from the supplier side (who often tried to show technical arrogance). In meetings called to resolve such conflicts, each group focused on emphasizing its own individual contribution and achievement instead of the joint contribution and achievement. The local team often complained that the software supplier counterparts delegated or referred to them tedious and awkward parts of the work while keeping preferred ones to themselves, during the customization. The team from the software supplier side complained about the mistrust from the local counterparts and continuous watch over to monitor their activities.

Looking back, the earlier approach followed to resolve the problem was also inappropriate or not targeted to addressing the real cause of the problem. Initially all attempts to resolve the problems were directed towards bringing together vendors and their project counterparts, discussing ways and means of correcting errors made in the execution of projects and mobilizing additional resources to meeting deadlines and delivery of milestone products. The discussions were often dominated by pointing fingers, externalizing shortcoming, putting others to blame. When such repeated efforts did not bring any difference in reality at all, the overall implementation approach was questioned. But this was discovered late. That was the time when use of alternative methods was explored. The approach presented below was one of the efforts in this new direction.

Using lessons learned from the application of the business process redesign approach developed by the author for use in related projects (discussed in detail in Chapter Four), the author took the initiative to remedy the project implementation at one of the branch offices of Organization C. The selected branch office was one of the largest in terms of customer base and business transaction. In particular, the researcher initiated a Friday afternoon workshop session with operatives to address problems related to reform implementation in general and the software implementation in particular. The workshops were conducted in the manner described in Chapter Six of this report. The author tried to explain the purpose of the workshops and demonstrated how to facilitate same in the first two consecutive sessions. As of the third workshop, users took over the facilitation and the author took the role of a participant. As required, the author intervened to coach the facilitators and guide the sessions, particularly reminding participants from time to time of the purpose of the workshop and the need for open and constructive argumentation. The author made it a point also to intervene whenever dominance by some participants or expression of authority by supervisor participants or insecurity from subordinates, etc. were observed. Gradually but progressively, the participation and enthusiasm of the community increased. A number of issues were raised, discussed and resolved in these sessions.

Each workshop session was designed to focus on issues related to a specific process agreed upon by the participants. At each workshop session, a facilitator and a scribe were elected by the participants to moderate and record the session deliberations respectively. At the end of each session, the facilitator, the scribe and the owner of the process discussed sat down together to analyze the outcomes of the workshop session. The results of the analysis were summarized in three main categories: issues raised and positions taken, plan of action to implement decisions, made and outstanding issues for consideration in setting agenda items for the next workshop session. Such a summary was circulated to participants (including the branch management) immediately within two days of the workshop session. Together with the summary of the proceedings, a reminder of the next workshop session that also requested participants to prepare themselves on the outstanding issues was circulated. Further details of workshop procedures applied (together with some modifications made during subsequent use of the procedures at the third project at Organization B with software professionals) are presented in Chapter Six of this report.

At higher levels, the branch management and the project office were required to actively and promptly address issues referred to them from the workshops. Actions taken and issues addressed were directly reported in the workshops, and this encouraged participants very much. This way, it was possible to take a number of actions that addressed most of the problems. A training program was also conducted based on the needs identified in these workshops. Some of the user complaints were resolved by making simple working arrangements. For instance, one of their complaints related to the longer time customers had to wait to get the attention of the front-office sales staff (who were the interfaces for the clients of Organization C). As the system was an integrated one, it enabled a front-office sales staff to serve customers of more than one service types. Usually, for instance, service requests, say for one of their services, were simpler and faster to process, as compared to another service type. So where customers of the two service types came together to the same sales person and the customer for the service type that takes longer approached the sales person first, the customer for the other (simpler) service had to wait until the sales person was done with the first customer. Some customers seriously complained about such waiting. In the discussions, it was learned that this did not require a software solution - a simple administrative arrangement was made to resolve this issue. Different desks were assigned for different services.

The workshop findings and the training sessions did not only help users to better understand the features and capabilities of the newly installed software and their use, but it also helped the developers to get feedback directly from the users, which helped them better understand the complaints and promptly address them to the satisfaction of the users. According to both developers and users, the developers and the project would have benefited a lot if such sessions had been instituted in the customization and implementation of the project from the outset. This could have avoided most of the problems around the software functionality. By the end of the second month, the news about the progress made at this branch circulated around all branches of Organization C and corrective actions taken at the branch where the approach was implemented helped resolve most of the bugs in the installed version. According to an assessment made afterwards, the exercise at the branch office did help to sustain the operation and support of the software. It also gave an opportunity for champions on the system to emerge from both the IT unit and customer service unit. There were an increasing number of people from both the technical and user sides that were determined to fix problems promptly and disseminate required information on the product features and capabilities as well as the project status and progress. This did help to avoid all kinds of misinformation and confusion around the software.

2.3 Chapter Closing

In this chapter, an attempt was made to describe the software development situation by sharing stories from selected cases that the author was actively involved with. The challenges faced as well as the success and failure factors were summarized as observations and lessons learned at the beginning of each case.

As indicated, on the technical front, within a span of two decades, we saw a shift from the use of programming language and operating systems features to the use of structured approaches and then object-oriented approaches in both teaching and professional practice. For the successful development of a usable system, in addition to the technical aspects, it was possible to witness the importance of: closer cooperation among key user representatives (domain experts), software engineers and end users, allotting time (by software engineers) to learn about and build knowledge on the application domain, and the use of operational prototypes. Among the factors that contributed to the difficulties experienced in the successful implementation of the project were: attempting to deploy large scale projects in one go; conducting systems design (business process redesign, to be more specific) and the related software development separately; introducing uncontrolled and incessant changes in requirements throughout the projects and reactive development in which developers continuously change directions; introducing new technology without proper and careful evaluation of the implications on the project; and overdependence on few specialists.

To obtain further information on the existing software development situation in general and to assess the situation with regard to methods use and related challenges faced by practitioners in the local setting in particular, a comprehensive survey was conducted as part of this research. The design, administration and findings of the survey are the subjects of the next chapter.

CHAPTER THREE

3. Software Practices and Emerging Demands in Ethiopia: a Survey

In the previous chapter, an attempt was made to provide a picture of the existing software development practices in the local setting by presenting selected case stories and related qualitative analysis. To get a more complete and up-to-date picture of the situation, further investigations were made to assess the situation through survey questionnaires and interviews conducted with the main actors in the field. While the work related to the development of a survey instrument and the findings from the survey conducted are reported in this chapter, further work done based on the findings is reported in subsequent chapters.

3.1 Survey Instrument and Participants for the Assessment of Practices

3.1.1 Questionnaire Design

Based on the analysis made on the case studies reported in the previous chapter and reviews of related works and the literature, a list of possible issues that were considered important for assessing the existing situation was first prepared. In the process of compiling the issues, consultations were also made to similar previous local attempts, including those by Rahel (2004), Ahmed (2006) and Getahun (2006). In these studies which were conducted in partial fulfillment of a Master's degree at Addis Ababa University (AAU) and College of Telecommunications and Information Technology (CTIT), attempts were made to develop discussion and survey instruments to collect information on issues related to software project failure in the local setting. Rahel particularly adopted the Capability Maturity Model (CMM) questionnaire recommendations, while the other two adopted qualitative methods commonly cited in the literature. Although these attempts might have served their purpose, they were not found to be thorough and fit for the sort of assessment under consideration in this research. Besides, the use of CMM questionnaire as is was considered a misfit to the local realities.

The issues compiled through the above process were further developed and enriched in the series of discussions held with colleagues and post-graduate students in the system development courses offered initially at AAU and then both at AAU and CTIT. These issues were then reworded as survey questions. The survey questions prepared were tried out by students in their coursework projects. As part of the discussion that followed the coursework assessment, the survey questions were further revised and focused. In particular, among the range of topics considered, issues related to the practical use, usability and appropriateness (affordability, availability, sustainability, etc.) of popular ('industry standard') software development methods/approaches, in the local settings were given more emphasis for immediate investigation. As part of the revision, and based on the feedback from the students and practitioners that took part in the coursework projects, the questions were revised by including texts that provided additional explanations and guidelines.

Based on the revised list of survey questions, three types of questionnaires were prepared for the purpose of this research: one for software companies (about 20 questions), one for the IT Departments in large organizations (about 16 questions) and one for software development professionals (about 35 questions). Samples of the questionnaires used are attached as Appendix 1 of this report.

The questionnaire for the companies was designed to obtain information on the company in respect of: company profile, customer profile, means of getting new customers, partnership arrangements, professional staff turnover, software development and project management processes, and guidelines being practiced, change management practice, project success rates and critical factors that need to be addressed to improve productivity and quality. Almost the same questionnaire was used for IT Departments with some adjustments required to make certain questions applicable to these organizations.

The questionnaire for professionals was designed to obtain information from the professionals in respect of: their formal education and training, qualification and software development experiences; means of upgrading software development skills, ongoing training and level of support provided by the employer in this connection; project workload and reporting mechanism; the extent of using software process models and methods and factors that affect the use; performance level and factors that impact performance; means and frequency of sharing information, experience and learning with colleagues (project staff); understanding the role of users, as well as purpose and mode of communication with users; the extent to which methods are factors for productivity, and the availability of guidelines, training and related support; challenging aspects of the development work; issues that need urgent intervention to improve usability of methods, productivity and quality of work; and important skills required for software development in the local settings.

In addition to the testing done as part of development process, the questionnaires were also further pilot tested with instructors and the new batch of graduate students (in the following semester) at Addis Ababa University as well as with some selected software practitioners in the industry. This was to check for appropriateness and clarity before actual use. Comments and input from the piloting were incorporated into the instruments before they were actually used.

3.1.2 Data Collection

Due to the lack of readily available empirical data on the software industry population in the country, selection of participants was guided by a list compiled for a previous study by Rahel (2004) and lists compiled by post-graduate students as part of a coursework to assess the extent of method use in software development by local software firms. A sort of purposive sampling technique was used in selecting participants from the lists. That is, attempts were made to ensure the inclusion of most of the IT Departments within large organizations that practiced in-house development of softwares. All private firms that were actively involved in the development and customization of medium to large software systems and projects were included. Those software companies whose major business was not software development or customization were excluded. In the selection of individual participants within these institutions, the researcher was guided mainly by references from the heads of the institutions. Here as well, care was taken not to exclude software engineers or project managers who had valuable experience on large projects that involved multiple stakeholders.

A total of 9 software firms, 20 IT Departments and 89 professionals were included in the survey. Applicable questionnaires were sent to all (hand delivered as all of the respondents were in Addis Ababa[6]) and to expedite the process, arrangements were made to physically collect the completed questionnaires. Each questionnaire was accompanied by an explanatory covering letter that stated the purpose of conducting the survey, and how the findings will be used. An overall response rate of 88%, 90% and 91% was obtained from software companies, IT Department and software development practitioners, respectively.

The interviews and discussions conducted for collecting data for this study included structured one-to-one interviews with selected software engineers, managers of IT Departments in selected organizations and software development companies. In particular, selected individuals from those that had already participated in the questionnaire survey were given more chance in the interviews to qualify their responses through providing explanations or examples.

As training and educational programs play very critical role in preparing and qualifying professionals, interviews were also conducted with senior instructors. With regard to the selection of instructors, only those specifically involved in teaching and advising students in the area of systems and software development (teaching such courses as systems analysis and design, software engineering, programming, database design, software project management, etc.) at AAU and CTIT were considered. In addition, group

[6] This is because at the time of the survey, Addis Ababa was the home of most of the software companies and corporate IT Departments, in fact the home of most of the technology related modern economic activities and services, as well as higher learning and professional education facilities in the country. Up until the federal structure was introduced after the downfall of the military government, most major business services operated from their bases in Addis Ababa with satellite offices in the regions. This was also mostly the case with computer related service businesses in general and software in particular. However, the opening of more professional schools and higher learning institutions in the regions, as well as the administrative and budget decentralization currently underway, the situation is likely to change in the foreseeable future.

discussions were also held with postgraduate students taking software engineering, system development and project management courses at AAU and CTIT; work group sessions were also held with the project staff at Organization B and with the project team under formation to engage in the development of eGovernment applications at the Ethiopian ICT Development Agency.

Interviews with the professionals were conducted either at their premises or a convenient place jointly arranged by the researcher and interviewees. As may be expected, the questions in the questionnaire were quantitative in nature, while those in the interviews were open-ended qualitative questions. All of the interviews were conducted by the researcher. Each interview session lasted on the average for one hour. The interviews and discussions were conducted in the Amharic or English languages depending on the preferences of the respondents. The interview and discussion notes which were jotted down during the sessions were then reviewed and translated (where appropriate) by the researcher for use in this study.

3.2 Discussion of Survey Results

In order to assess the extent to which the use of methods influenced existing practices and to study the conditions that will enable methods to enhance production of better quality products faster, respondents were asked to provide information on a number of issues including the following (for details, refer to the sample questionnaires attached as the end of this report):

- on the state of method use in the respondents' communities (including the attitudes of practitioners towards using popular methods or those adopted/adapted in-house);
- on the respondents' views regarding operational methods and guidelines in local firms;
- on the factors that had an impact on the system development process, success or failure of the system development projects;

- on the extent to which practitioners are well-equipped to use methods; on the most urgent interventions required to improve performance; and so on.

The findings from the survey as well as the individual and group discussions are presented in the remainder of this Chapter. In order to render meaningful discussion in this and subsequent chapters around the research questions, they are presented by describing the views of respondents followed by related interpretations made by the researcher, each organized along common themes, categories and perspectives formulated grounded on the data collected.

3.2.1 Profiles of Respondents

Of the professionals who participated in the survey, about 84% hold a bachelor's degree and about 11% a master's degree. Table 1 & Table 2 present summaries of field of study and years of experience of the professionals who participated in the survey.

Field of Study	**% of respondents**
Computer Science	40%
Information Systems	22%
Software Engineering	11%
Management Information systems	7%
Computer Engineering	6%
Other fields of study	14%

Table 1: Profiles of respondents by field of study

Year of Experience	**% of respondents**
Above 5 years	25%
Between 2 and 5	38%
Less than two years	37%

Table 2: Profiles of respondents by Year of Experience

As can be seen, most of the practitioners surveyed technically qualified but seem to be less experienced (the majority with less than five years of experience).

3.2.2 Types of Development Projects and Organizational Environment

The local software industry activity seems to be dominated for the moment by projects that support local needs and requirements in government and public service organizations. That is, the software development activities are dominated by outsourced projects that deal with custom-building of application software for specific client organizations. Projects targeting the development of generic software packages for commercial purposes are very minimal (if not totally absent). In some of the cases where there are similarities in functionality, the software written for one client organization is modified to create another version for the other client at source code level. Even where customization of commercially available packages is involved, according to the respondents, the work involved is so huge that they consider the effort as full-scale development. Among the factors that make the customization work huge are the peculiarities of the local realities, particularly the enormous amount of modifications required to accommodate the workarounds introduced in the business processes to handle exceptional and non-standard cases (these partly result from the lack of continuous improvement of the business processes).

As gathered from responses of software companies, there is a predominance of software development projects (about 50%) from the government sector. The maintenance of full-fledged IT departments within public organizations as well as the prevalence of software development projects that are managed by these departments further strengthens this observation (i.e., the huge demands in the government sector). On the other hand, most of the existing business processes in the government institutions are reported to be extremely backward and outdated. For that matter, by policy directives, currently all government and public service institutions are required to undergo major service reform that includes the reengineering of their business processes with the ultimate goal of institutionalizing best practices in their respective area of business. According to the discussions with practitioners and the researcher's own experience, most of the reform initiatives involve major restructuring of the organizations, management reform and overhauling of the processes. What is more, according to respondents from software companies, with the presence of full-fledged IT Departments in large organizations that intend to establish the required capacity for developing software in-house, the decision to use own resources to develop required software, or to outsource the development or purchase commercially available package has not been straightforward. Such is the environment within which the software projects are being carried out (more will be said on the effect of the reform initiative on the software processes later in this chapter).

3.2.3 Use of Methods

Modeling Tools, Process Models and Programming Languages

The following is a summary table presenting the modeling and programming techniques, development tools and process models used by the respondents.

Modeling Techniques		**Modeling Tools**		**Process Models**		**Programming Languages**	
Object oriented	43%	UML	42%	Code and fix	19%	Visual Basic	63%
Structured	26%	Rational Rose	23%	Water fall	13%	C-sharp	30%
				Requirement analysis and coding 49%		Database Lang.	30%
						Java	10%

Table 3: Modeling Techniques and programming languages used by companies

As can be observed from the above table, most of the respondents used object-oriented analysis and design techniques as compared to structured methods. It can also be observed that most used UML as modeling tool. When it comes to development languages, Visual Basic seemed to be very popular, followed by C-Sharp and database languages. As observed from the survey, the use of Java is just emerging.

According to the survey results, most of the companies were observed not to use any ‘standard’ method in the actual conduct of the software development projects. A commonly employed approach involved the use of cyclical requirements gathering and then programming practices, with a strong tendency to rely on in-house-developed practices rather than industry standard guidelines. The iterative processes practiced do not involve such practices commonly suggested in the industry as risk assessment and reflection, and the production of the software through increments. About 49% of the respondents indicated that they use some sort of iterative cycles involving requirement analysis, programming and testing solutions. This is in marked contrast with the 13% who have indicated they use the classic sequential life cycle mode (waterfall), and the 19% who said they simple follow a sort of code-and-fix method.

Respondents from IT Departments and software companies were also asked to comment on the extent of institutionalization of methods in their organization. By institutionalization of methods in these firms was meant the extent to which the use of industry standard or in-house developed methods have been accepted, integrated in organizational routines and practically followed in executing project activities. More specifically, this relates to the availability of guidelines in accessible format on such aspects as: contract negotiation and approval of project resources, documentation, progress tracking and management, change control, training and experience sharing, etc. About 67% of the IT Department and 29% of the software companies indicated that there were no formally documented policy and procedures to guide these processes. On the other hand, in response to items 32 and 33 on the questionnaire for professionals, about 49% of the professionals identified absence of guidelines on method use as critical limiting factors to carry out their tasks effectively, and about 57% indicated adoption of guidelines and standards as an area that needs urgent intervention.

Project Management

From the survey conducted and discussions held with practitioners, employment of a disciplined project management approach does not seem to be widely practiced. This is evident from the responses given to items 32, 33 and 34 on the questionnaire filled by professionals. While the majority of the respondents (professionals, software firms and IT departments) considered the use of methods as crucial for system development, they also believed that there is generally a lack of skill in the effective use of available methods and more so in the area of soft skills (where some have even reported the total absence or lack of awareness in this area). With regard to the importance of methods and disciplined approach, about 71% of the software companies and 67% of the IT Departments identified the use of standard methods as most important to produce quality software on time. On the other hand, however, in responding to item 20 on the questionnaire for companies, almost 86% of the software companies identified the introduction and practice of disciplined software development project management as one of the most important factors to deliver quality products on time and within budget. From the part of professionals, most (53% of the professionals) indicated the introduction of disciplined project management as an area that needs urgent intervention.

Training

According to the respondents, a number of factors might have contributed to the absence of effective use of popular methods or customized versions of these methods. Among the factors are: lack of training (as indicated by 19% of the professionals), the limited scope for applicability of the methods which are appropriate for "well defined" projects and problems (as indicated by 20% of the professionals).

In particular, the training requirements were expressed in many ways by the professionals: 46% indicated lack of training as the critical limiting factor to carry out their tasks effectively, 65% indicated skill upgrading as an area that needed urgent intervention; while about 57% of the software companies and about 89% of the IT Departments identified skill upgrading training as a critical requirements to produce quality products on time.

One obvious way in which to address the perceived shortcomings in software processes and methods is through training and education. Respondents were therefore asked related questions, particularly to comment on the quality and relevance of education and training provided by institutions of higher learning that supply graduates for the software companies and IT Departments. The results indicated a fairly poor impression of abilities of universities to produce the kind of competencies that software firms require of their graduates.

As observed during interview sessions, most senior practitioners and employers in the industry believed that the teaching approaches adopted in higher learning institutions, particularly the use of textbook case studies and/or simulated project exercises for practicum in teaching system development methods bear little resemblance to the problems encountered in real-life projects. Lack of real-life project and industry experience on the part of the instructors was also mentioned as an attributing factor to this. In similar discussions with instructors and selected students, the discussants expressed their views that the exercises encountered in the classrooms provide sound introduction to the methods by way of describing them and demonstrating their application in so far as developing specific skills in system design and/or programming are concerned. They also expressed that the classroom cases have limitations in terms of addressing the complexities of real world systems and that real-life cases would positively contribute to address the shortfalls in the effective use of methods in practice. However, while appreciating the benefit from incorporating a real life based project in the courses particularly to develop such vital soft skills as interpersonal communication and teamwork, the academicians questioned the organizational feasibility and sustainability of such arrangements in real-life. There were also some who explained the complaints from the employers in terms of lack of clarity or confusion in making the distinction between training and education – these academicians believed that such practice related skills are more appropriately addressed through special training programs designed and organized by the industry rather than through formal educational programs in higher learning institutions.

3.2.4 Other Performance Inhibiting Factors

Both professional and company respondents agree on the delays often experienced in project execution. When asked about the reasons for such delays (item 19 on the questionnaire for professionals), they indicated that, changing requirements and difficulty in getting users to clearly articulate their requirements as most influential factors. Among the other causes indicated by professionals were lack of properly defined roles and responsibilities and high workload. The following table summarizes the responses of professionals with regard to the main reasons for delays in software development projects.

Reasons for Delay	**% of professionals**
Changing requirements	54%
Difficulty of users to clearly articulate their requirements	48%
Lack of properly defined roles and responsibilities	25%
Work overload	24%

Table 4a: Professionals view on reasons for delays in project execution

From the responses of software companies (item 17 for questionnaires for software companies), poor planning because of unclear or incomplete requirements; high project staff turnover and lack of cooperation from users were identified as the main causes for project delay. The following table presents a summary of the responses.

Reasons for Delay	**% of Software Companies**
Poor planning because of unclear or incomplete requirements	50%
High project staff turnover	40%
Lack of cooperation from users	43%

Table 4b: Companies' view on reasons for delays in project execution

However, although as shown above 43% of the software companies attribute the lack of cooperation from users as one of the main causes of project delay, only 22% of the IT Departments confirmed this. This is to be expected as the latter group (the IT unit) belongs to the same organization as users and thus has a better position, access and working relationship to improve cooperation from users. While the software companies did not at all point out 'lack of proper progress monitoring and control' as one of the major causes, 44% of the IT units indicated this as one of the main causes for the delays experienced.

In relation to the critical factors that negatively impacted their performance, absence of guidelines and methods and lack of proper leadership and management were indicated as the main ones by most of the professionals. Lack of training and properly defined roles and responsibilities were also among the factors indicated equally critical by the professionals. The following is a summary of the responses in this connection.

Factors that negatively affected performance	**% of Software Companies**
Absence of guidelines and methods	49%
Lack of proper leadership and management	47%
Lack of training	46%
Lack of properly defined roles and responsibilities	43%

Table 5: Factors that negatively affected performance

What follows is a further discussion on some of the above performance inhibiting factors, particularly: work overloads & lack of properly defined responsibilities, deficiency in organization and documentation skills, lack of user cooperation, communication problem and changing requirements.

Work Overloads & Lack of Properly Defined Responsibilities

It was observed from the responses that software professionals were increasingly unhappy with the way their work was structured and the workplace pressures they faced. According to the respondents, they worked harder, spent more time at work, but saw little or incommensurate results. The survey results showed that the development team members were overloaded: about 51% indicated that they were assigned to 3 or more projects at a time, while 16% indicated that they were assigned to 2 projects at a time, and 30% indicated that they were assigned to only on 1 project (refer to Item 4 of the questionnaire for professionals). This situation was further aggravated by the very high turnover of project staff. According to the survey results, over the last five years as from the time of the survey, about 49% of project staff left IT Departments. Out of those who left, 71% left without completing project assignments. In about the same time, 30% of the professionals left from the software companies, mostly without completing their project assignments.

According to the respondents, most of the project work is organized in teams, and team members were drawn mainly from the developer side and mostly composed of technical people. It was found out that, across all project staff there was lack of organizational competence resulting in extreme shortage of candidates to assume project management responsibilities. This was also among the factors for the commonly observed phenomenon where an individual is assigned to manage multiple projects. In about 43% of the cases, the software unit heads in the software companies acted as project managers on top of their responsibilities in the technical team.

In relation to roles and responsibilities, about 43% of the professionals indicated lack of properly defined roles and responsibilities as one of the critical factors that negatively impacted their performance. It was observed during interview sessions that users also shared this concern.

In response to a report by an expert who was called upon to facilitate discussion to resolve a conflict between the in-house IT experts and the consulting team, a project sponsor (a senior executive in the client organization) witnessed the situation as follows.

> "… I fully agree on the points enumerated as prime causes for the problems encountered so far in the progress of the project. Probably one additional point is the lack of clear and neat definition of roles, tasks and assignments in the first place to the best understanding of both parties probably because the situation in the early stages of the project did not call for such exercise. …. Once again a point leading to misunderstanding is the lack of clear definition of the role of the consulting team, i.e., a decision making vs. a consulting role. I feel the order is that the Consultants provide their professional advice and recommendations and the owner decides what he likes".

Deficiency in Organizational and Documentation Skills

From the discussion conducted and the researcher's own experience of working with some of the project teams, most of the professionals seemed to have good technical skills particularly when it comes to programming. What is striking is the dearth of basic knowledge and skill in organization and communication even among those that are said to have more experience and to exhibit better skills in documentation. To this end, questions were also asked about the challenging aspects of their work. The results presented quite a mixed picture but there was some justification in drawing a conclusion that more technical tasks like programming were taken as relatively less problematic. In general above 50% of professional respondents indicated that more managerial and organizational tasks such as project planning, requirements gathering, coordination and communication, are relatively more challenging.

Discussions were also initiated to find out the skills and attitudes of the practitioners towards documentation. From the discussions held, it was possible to observe that senior practitioners that joined the software development process from other fields and those with prior management orientation emphasized the observance of discipline and use of extensive documentation. On the other hand, the new graduates were classified by employers (software companies or IT departments) in two categories. In one category are those that are not only inclined to and are good in hard core programming (particularly with visual basic, c-sharp and java) and not only disinclined but incapable of writing reports and/or documentation (particularly those graduates of computer science). In the other category are those that have better skills in documenting user requirements and general design as well as application development using database languages but they are less skilled in hard core programming (most of the graduates from information systems programs).

In general, what is perhaps encouraging in this connection is the increasing recognition among practitioners of the importance of human and organizational skills and their implications in software development. This is gradually getting foothold among both users and developers as of recent. This appreciation, however, did not yet manifest itself in the skill development requirements and training programs. Despite the deficiency in the non-technical skills, the focus of professional training programs provided were still predominantly in technical areas as summarized in the following table.

Professional Training Programs	**% of Focus**
Programming	60%
Database System	40%
Maintenance and Trouble shooting	5%
System Development	3%
Project Management	1%

Table 6: Focus of professional Training Programs

Lack of User Cooperation

According to the survey results, 70% of professionals expect the contribution and involvement of users in the requirements gathering stage, about 25% expect in the implementation stage. In response to item 33 of the questionnaire for professionals, it was found out that 53% identified the need to establish mutual understanding with users as an area that needs urgent intervention. Moreover, 86% of the software companies and 100% of the IT Departments identified the involvement of user representatives in the development team as the most important requirement to produce a quality product on time.

On the other hand, about 43% of the software companies indicated the lack of cooperation from users as one of the main causes for their poor performance. When the issue of lack of cooperation from users was probed in conversation with project managers, software engineers and educators, the lack of a common language to represent and communicate users' thoughts was usually the first factor mentioned as being the cause. However, many also indicated that the problem was deeper than this and had to do more with motivation and trust in project goals and objectives. Professionals believed that most users communicate well, once they clearly understand and are convinced of the objectives of the project, or once they get recognition for or are provided with some incentives to compensate for the extra effort that they put in the project, or once they know that the software has something to offer to simplify their working life (adds value to their work and career). Some software practitioners who were interviewed made frequent references to differences in work cultures, noting that when dealing directly with users, it was necessary to establish a relationship over time, with repeated personal interaction, in order to create the kind of mutual understanding and cooperation required to effectively and efficiently carry out their project activities.

Communication Problem

Probing further into the communication practices among the development team members and between the development team and users, what has come out strongly is the practice of sharing project related information, experience and learning through informal meetings and discussions. As can be seen from the responses of professionals to items 20, 21, 26 and 27 on the questionnaire for software professionals, 82% of the communication within the development team was made through informal means (with only 23% through regular meetings) and 70% of the communication between the development team and users was made through informal means (with only 28% through regular meetings).

In this connection, it is relevant to note that attempts made to find published materials in the form of project journals/logs or meeting records that were used in the communication were unsuccessful. It was learnt that documenting design deliberations, issue resolutions etc. for the purpose of learning and communication was not commonly practiced among the professionals. On project related matters, team members were not routinely and formally informed of the status of a project; were not reminded of approaching scheduled milestones; were not provided with feedback on how well the project is doing in the eyes of colleagues and stakeholders; were not informed about outstanding issues to be jointly addressed, etc. Even where it was indicated that there were regular meetings and reports, discussions with selected professionals revealed that such meetings were held to respond to certain ad-hoc inquiries from management, and the information compiled in such meetings are supplied up the management chain. This omission of bi-directional and horizontal information exchange and communication, coupled with the lack of participation in the planning (see below) of the project has no doubt discouraged ownership of the process by the team and being part of a solution. Most of the members of the development team became aware of the project relatively late (after the signing of contracts). As observed from the response to item 10 on the questionnaire for professionals, about 24% of the project staff expressed that they were not involved in the planning of any of the projects they were working on. Without such early exposure, involvement, information provision and formal coordination and follow up mechanisms (formal project meetings, formal client meetings, based on formal documentation, etc.), it may be difficult to keep team members stay focused on and committed to project goals and objectives. As can be seen from the responses of professionals to items 31 and 32 of the questionnaire, over 50% of the respondents identified coordination with the team and communication with users as challenging, with 38% identifying the lack of proper communication and collaboration as critical limiting factor to carry out their tasks effectively.

Still closer analysis of the situation indicates that most projects do not seem to have a clearly defined and communicated plan, where progress is quantifiable and regularly monitored and controlled. Metrics that provide the information necessary to assess project progress were not properly established and used. The process of feedback was unavailable. There were no clearly defined and communicated processes that cover the various activities of the development project. This information vacuum severely limited the abilities of the project staff to control and mitigate the risks associated with a project. Obviously, in the absence of a disciplined approach, a defined process documentation and management, project success solely relied on the skills, talent and efforts of individual members of the project team. This coupled with the high turn over of skilled personnel (who were in a very short supply in the market) did put most projects in a very vulnerable position.

Changing Requirements

With regard to indicating the scale of change in requirements over the course of the project period (in particular, in terms of the degree of variation between the requirements as stated in the original tender document or assignment brief and the actual implementation), 57% of the software companies indicated the existence of a major variation. About 43% of the respondents attributed this variation to the inability of users to clearly articulate the requirements upfront and 28% to the changes that took place in the organization since the commencement of the project (refer to item 18 on the questionnaire for software companies). When asked to comment on the impact of the changing requirements in an interview that followed the questionnaire survey, some members of the software companies stated that software products developed and delivered based on the initial requirements specified by the users, had to be extensively revised over and over again during use. Professional respondents described the often difficult time they had experienced in convincing and motivating users to actively and seriously involve themselves in the development process as early as possible. Most comments pertained to the difficulty to motivate users or their representatives to actively participate in earlier phases of the project. Interestingly enough, user involvement during the use phase is reported to be very high. In this connection, one senior developer stated during an interview:

> "I have 10+ years of software development experience and involvement in more than five major projects. In those projects where we claim success, we have developed a completely new version of the software once we started work with the users as part of the testing phase of our originally delivered version (supposedly designed per the requirements forwarded as part of the contract). What I have learnt from experience is unless you develop and deliver a working version of the product to the users and condition them to use by some form of arrangement (be it by management decision to test and accept the system developed by the software group or otherwise), it is not possible to get users take the system seriously and get valuable feedback and the actual/real (rather than the stated) requirements. In my experience, users (at least those that I have worked with) tend to cooperate and articulate their requirements when they know that the deployment and use of the system is for real (as most projects rarely make it to that level for one reason or another). It is only at this stage that they become interested to proactively work in real partnership with the development team towards the success of the project".

This observation was put across to others and they were asked to reflect on this observation and indicate whether they agree or disagree to it. Surprisingly, looking back on their past experience, most professionals shared and confirmed the observation. They stated that, in most of the projects that were made operational, the actual development took place at the users' site under the guise of testing the initial version delivered per the contract terms. They stressed the fact that much of their effort went into the redesign of the software rather than the often talked about concern at this stage - fixing bugs. According to the professionals, the redesign was necessary due mostly to the failure of the delivered system to meet users' expectations (resulting from the misunderstanding of workplace realities and workarounds by developers, and attempts to base the design on requirements stated in reports prepared for this purpose and incorporation of features and capabilities borrowed from other similar systems). The redesign was necessary partly due to changes in the organizational processes resulting from the reform initiative which more or less has become commonplace in all government and public organizations.

Furthermore, some professionals have also reported how users or their key representatives valued the close working relationship during this phase – in their language, this gave users the feeling that 'the professionals are there to help them out'. According to the professionals, it was this feeling of 'being there' for users that for the most part motivated them and provided the developers with a unique opportunity to establish the necessary rapport with the users, and learn more about the application domain from the users. This in turn helped to create effective communication and mutual understanding required for the successful completion of their project. Such degree of closeness, physically, organizationally and culturally, between collaborating parties was said to facilitate more frequent and often intensive interactions that were necessary for the successful development and use of the newly introduced system.

3.3 Emerging Demands

In the discussions that were held with selected practitioners as a follow up to the questionnaire survey, software development projects related to eGovernment applications were frequently cited as one of the representatives of upcoming/emerging challenging projects in the local context. Most of the projects were being outsourced at the time of conducting this research. Accordingly, an attempt was made to look into such projects to the extent of further addressing the research questions under investigation in this research. The following is a brief account on the eGovernment cases reviewed as part of this effort.

3.3.1 The eGovernment Initiative

According to the discussions with representatives of EICTDA (the national coordinating body for the program), the main objective of the eGovernment program is to provide IT-based support systems that improve both efficiency and information management within the various ministries and public agencies of the federal and regional governments. In particular, the program aims at introducing integrated information systems within the various government ministries and agencies to:

- improve the internal efficiency of the ministries/agencies in running day-to-day operations;
- improve access to essential information including inter-ministry/agency access to essential working information;
- enhance the transparency of interactions between the ministries/agencies; etc.

The program is also designed in a manner that can support the civil service reform programs already underway in these institutions for the purpose of overhauling the processes, structures and human resources.

As part of the eGovernment program, a number of large software development projects were being tendered out. For instance, one big project launched at the Ministry of Finance and Economic Development (MoFED) was the Integrated Financial Management System (IFMS). According to the project manager, this project is expected to provide functionality to enable all government entities (ministries, agencies, regions, etc.) undertake budgeting, accounting and financial management. The implementation strategy adopted was a turnkey implementation by a single supplier selected based on an international competitive bidding. As part of this engagement, the selected supplier takes the responsibility for the delivery of all the components of the IFMS including delivery of the application software, project management and training, hardware components, and provision of support after the completion of the implementation. The full implementation of the project was planned in three stages. According to the project manager, "IFMS implementation would be viewed as an institutional and change management system that would be used as a tool to realize the business process reengineering activities of the country and to improve the skills of Government staff".

The project at MoFED is but an example. There are similar other eGovernment systems planned for execution by each government ministry. Looking closely at the specifications, while most of the business application requirements, for instance, in the general-business applications category have common functionality, there are also variations: in their local/regional language interface requirements (due to variations in working languages per region); in the size of population to be served and related transactions; in the application of local government laws and regulations (variations in applicable taxes for instance); etc.

Most of the eGovernment applications are to be installed and run in branch offices and/or organizational units that are geographically scattered across the country. Applications that run across multiple agencies have features and functionalities that may be organized as shared services across the agencies. Most of the eGovernment application programs are required to share common network infrastructure[7], as well as reference databases, rules and regulations applicable to the various sectors of health, education, etc. Implementation of the various applications may require massive conversion and migration of both applications and contents/data. Almost all the eGovernment applications require business process reengineering and reform as part of the system development activity (more specifically, as a front-end process to the software development). Such requirements were explicitly stated in the tender documents as in the case of IFMS above.

According to practitioners, most of the eGovernment software development projects which are basically large and complex may be difficult to realize given the environmental conditions within the government organizations, particularly the level of readiness because of extremely backward and bureaucratic processes and pessimistic attitudes of

[7] There was a newly deployed network infrastructure that was made up of high capacity broadband network capable of supporting full-fledged multimedia voice, data and internet applications and services up to Woreda (district) level and narrowband network capable of supporting voice and narrowband data and Internet applications and services to Kebele (residential area) level, and nation wide.

users towards change in general and automation in particular. Other related studies (Heeks, 2003) also indicated that although such programs can make a valuable contribution to development, at present, the majority of such projects fail either totally or partially. The oversize gaps between project design and on-the-ground reality was cited as one of the main causes of failure.

While there is no space here to make an extended review of the eGovernment related software development projects, two aspects that commonly feature in most of these projects (and shared by other similar projects in the local setting) are worth noting to serve as a background to the strategies explored in subsequent chapters for tackling the local situation. Particularly, the undersupplied tendering process and inadequate project organization; and the lack of actual integration between the business process reform activity of the service reform project, network infrastructure building project and software application development project. Each of these is briefly discussed below.

3.3.2 Undersupplied Tendering Process and Inadequate Project Organization

Looking closely at the history of local software development experiences, we observe that a common practice in the past was for organizations to have their software systems developed in-house by own IT staff. Since recent times, however, the tendency in most organizations has been to outsource their system development projects to external entities based on a competitive bidding process. In the outsourcing processes, the tender document, among others, is a basic planning document that is expected to at least specify the project's goals, scope, solution required, time constraint, etc. The information in the tender document is expected at the very least to enable potential bidders to prepare responsive proposals. It is also customary to include information on the criteria to be used for analysing proposals from vendors to select the most responsive one. Where there are existing standards and environmental factors to be considered in the course of preparing proposals, such details are also explicitly stated in the tender document. Usually the tender document and the bid analysis report will be used to prepare a detailed project contract that guides the development and successful implementation of the software.

According to the findings following the survey, discussions and the researcher's own experience, most of the tender documents floated in our case were very brief and do not seem to include enough information to prepare responsive proposals, even for such large projects as those in the eGovernment program. Most of the project details seemed to be left open for bidders' interpretations. For instance, it was not uncommon to find bid documents that demanded vendors to perform business process redesign and then develop the required software, leaving decision related to costing and scheduling of the software development project entirely to vendors' considered opinions. Such lack of clarity gave way to all kinds of misinformation and misunderstanding between the parties (users, consultants, implementers) throughout the life time of the project. According to some bidders that were involved in the survey, misunderstanding of the requirements coupled with the strong desire on the part of bidders to win the tender often led them to extremely underestimate the complexity of the projects and to submit low-priced offers. Where the requirements were generic and brief, vendors felt that the most deciding factor in the analysis of vendor proposals particularly to differentiate one from the other would be financial rather than technical, leading to the submission of low-priced offers by vendors to win the tenders. As expected, such strategies mostly worked for the vendors in terms of getting the contract awarded.

In related discussions, there were complaints from clients/users that vendors usually include CVs of high-caliber and experienced individuals in their proposal just for the purpose of winning the bids and often bring in less-qualified and inexperienced people for the actual conduct of the project. This was partly because they could not afford to employ the experienced ones with the low price with which they won the tender. While partly sharing this view, vendors provided another reason for such eventuality. They attributed this to the overly extended tendering process which mostly went beyond the scheduled availability/commitment of the professionals identified for engagement in their original proposal. According to the survey results, almost all of the software houses (86%) complained about the very lengthy tendering process which often went beyond the validity of offers made by vendors.

On the other hand, any system development project itself is an organization. It is a collective undertaking that involves many persons and groups requiring cooperation, timeliness and management. To be successful, such projects (particularly the large ones) have to create and maintain temporary organizational networks linking users to system developers, decision makers to workers, and consultants to clients. Therefore, in handling such projects, both technical and organizational competences are key factors. All this notwithstanding, such organizational factors were often ignored or not adequately addressed in both tender documents and vendor proposals for one reason or another.

According to the facts on the ground, compromising quality or aborting projects due to considerable cost and time overruns, inability to engage the services of qualified professionals for the entire duration of the projects due mainly to insufficient funds, and lack of organizational competence to manage the project in a disciplined manner were all commonplace in the local setting. These were also evident from the cases discussed in Chapter Two of this report. In addition to these cases, a number of others were cited in the discussions that were held with the practitioners. The cases ranged from those projects that were implemented without producing results that match the original intentions (the case of a student support system in a multi-campus and multi-faculty institution of higher learning) to those that miserably failed on all cost, time and quality/performance counts (the case of a national public body).

One interesting case cited related to the case of an insurance company. This was a company with a fairly large business operation, but different from Organization B discussed in Chapter Two. The company experienced a number of failures in its automation efforts over the last couple of years. The first related to the acquisition of application software to be supplied by a foreign software company. To that effect, an agreement was signed between the companies. Based on the agreement, the supplier worked for several months with the users to define the specific requirements as well as the related features and capabilities of the required software package. Prototypes (more specifically, throwaway prototypes) were used as a vehicle for this purpose. Eventually, after working in this manner for almost a year, a decision was made to freeze the requirements and proceed to the customization of the software package based on the requirements defined thus far. In parallel, per the advice of the supplier based on the requirements of the software under customization, preparations were made in terms of revising work procedures and converting data from the in-house system to the new system under development. After joint decisions were made on a cut-off date to fully migrate to the new systems after working for almost eighteen months, the supplier was unable to deliver an operational version of the software due mainly to the inability to retain (or replace) key professionals of the development team that left the project. At last, this resulted in the cancellation of the project. This was followed by another project initiative to purchase and customize a commercially available and proven product. As a result, another (probably more expensive pricewise) software was purchased after a lengthy procurement process that went on for years. The installation and customization took another couple of years. Worse still, it was learnt that, due to difficulties experienced in the use and maintenance of the system (mostly to do with high maintenance and upgrade cost demanded by the foreign supplier), a decision was made to discontinue using the software. As a result another procurement process (for the third time) was initiated. At the time of writing this report, the third project was under implementation.

Taken together, while it may be difficult to totally avoid cost overrun, time slippage and unsatisfied demands, particularly with large projects, adequate emphasis and care on the tendering process and project organization could have no doubt helped to keep the critical success factors within bounds. The survey results as well as discussions and reviews at various levels showed deficiency in the tendering process and organizational factors to be among the exceedingly prevalent causes for the failures, particularly in the case of projects that involved outsourcing to foreign companies.

3.3.3 Integration of Software Development with Organizational Reform

Business activities at all levels are information-intensive, so much so that information has become a key organizational resource that is central to all business processes and functions. The information system that manages such information within an organization is increasingly being recognized as the nervous system of the business. As such, all organizational-level business process reform initiatives such as the ones underway locally in the eGovernment programs, no doubt directly affect the underlying information system. Accordingly, to deliver reform objectives in full, reform initiatives within these organizations often consider reengineering of the underlying information system (both infrastructure and applications) very seriously right from the outset. There is increasing awareness and understanding that where process reform, infrastructure building and software development efforts were undertaken in isolation (independently), the projects usually end up in failure when it comes to delivering reform objectives in full.

In the case of the eGovernment program under reference, for instance, the network infrastructure planned to support the systems was already deployed. According to the findings of the survey and the discussions conducted at various levels, business process reform initiatives and software development efforts within the same organization and relating to the same application were being conducted more or less incoherently. The following are some examples from the cases cited during the discussions.

- The first case is related to an application software development project in a public service organization. As part of the implementation of a software project, an attempt was made to redesign the business process to be supported by the software. That is, the business process had to be redesigned in accordance with the requirements of the software package which was considered to include best practices from the industry. For this purpose, a business process expert with a good practical knowledge of the application domain had to join the software development team to interface and work with both users and the development team. According to the project staff, although the project required extra effort and tough decisions that at times required the intervention of the most senior executives, the business process redesign and the software deployment were completed and the system was made operational. In parallel with the software deployment, however, there was a reform project going on within the organization. In line with the national guideline, the reform program involved strategic planning, business process reengineering and the introduction of result-oriented performance management, planned for execution in that order. By the time the software was made operational, the business process reengineering work was in execution. This reengineering process challenged the business process newly redesigned as part of the software deployment. At the time of the survey for this research, serious discussions were underway within the organization as to how to proceed in reengineering the business process with the users divided in two groups. One group insisted that the business process be overhauled again as part of the business reform. While the other group strongly argued that the business process redesign was already done and redoing it would add no value but result in service disruption and incurring unjustified expenses for software revision. The issue which was referred to the top management of the organization was not resolved up until the writing of this report.

- The second case cited related to a project that aimed at automation student activities in one higher learning institutions. Like the case described above, the software development effort was not integrated with the business process reform activities. The reform involved, among others, the restructuring of the Registrar's Office that gave the various academic units more autonomy in handling student cases and the maintenance of academic records as well as the introduction of cost-sharing scheme whereby students are given loan to pay fees for later repayment (after graduation). The software development which took place at about the same time did not provide for these new requirements (as this did not feature in the original requirements specifications). This happened to be the point of serious dissatisfaction and dispute between the client and the software house on the one hand and the central Registrar's Office and the various academic offices on the other. As a result, the software although installed was not fully functional. The management of the institution demanded a complete assessment of the situation with recommendation on a way forward (even if it meant replacing the already deployed system altogether).
- The case of Organization C discussed earlier in Chapter Two is also another example. In this case, to some extent an attempt was made to redesign the business process prior to the acquisition of the software (and the redesigned process was used in the process of acquiring the software). However, the business

process redesign, it was later discovered during the customization process, did not fully consider best industry practices. The business process redesign did not also properly consider the integration of processes across product lines.

As part of this research, attempts were made to take up these issues with concerned bodies for the eGovernment applications. In the process, it was possible to learn that based on a series of assessments made at various levels on this particular issue, a decision was made to take corrective measures for forthcoming projects. As a result, newly floated tender documents were observed to indicate the need to do business process redesign prior to the development of the software required. Yet, the level of details provided was inadequate; requirements related to actual integration with the software design process were not addressed; special requirements to assess vender competence in this connection were missing.

Still to deliver reform objectives in full and in a sustained manner, in parallel with the effort to integrate business process reform, network infrastructure deployment and application software design activities, we need to seriously consider the human aspect. In other words, over emphasis on technology or technology-focused reform initiatives and process reforms that ignore the skills, information needs and communication preferences of users and relevant stakeholders are bound to fail.

As with the case of Organization C discussed in Chapter Two of this report, the implementation of large scale IT projects and the introduction of modern information systems in organizations often involve and imply changes in the user organization, sometimes resulting in turbulence and instability. To avoid negative implications and maximize benefit from such projects, it is important that the development and implementation processes be accompanied with appropriate training and change management efforts (Kimaro and Nhampossa, 2005).

3.3.4 Organizational Embedding and Sustainability

Smooth and successful development and deployment of the software systems are necessary, but not sufficient, conditions to realize benefits from the reform programs.

Sustainability in the sense of the organizational ability to effectively use and maintain systems once developed and deployed, is another important factor that deserves attention. In operational terms this relates to issues of embedding the system into the organization. This could be checked in terms of assessing to what extent the system developed is effectively used and appropriate to the organization and its users. It could be checked if the system developed is flexible enough to be adapted to the changing needs of the users and the organization as a whole over time. Whether or not there is adequate local capacity and resource to translate changing needs to system design and development efforts could also be checked (Kimaro and Nhampossa, 2005).

In short, over and above changes at the technical and infrastructure levels, the introduction of the new system into the organization particularly requires the cultivation and institutionalization of a new kind of culture and ways of doing things that are associated with the newly introduced system. The introduction may include revisions in work activities, roles and responsibilities, structures, ways of gathering, processing, reporting and using information, etc., in the existing organizational routines. Successful use and maintenance of the newly introduced system then requires that these be understood/observed and adjustments made by all actors continuously.

All this notwithstanding, in the existing realities of software development practices in most of the cases examined, the sustainability aspects were not properly addressed – one could even argue, based on the facts on the ground, that these aspects were largely ignored. There was a tendency among the people involved in the software development that this aspect particularly the use and organizational embedding related issues and requirements thereof are subjects of organizational change management. This is the case even when, as indicated earlier, the developers themselves experience major involvement in this phase of the project to ensure acceptance of the system developed.

3.4 Chapter Closing

In the previous chapter, an attempt was made to describe the software development situation by documenting challenges faced by practitioners, as well as project success and failure factors, based on selected real-life project cases. This chapter tried to describe the software development situation based on the opinions surveyed using questionnaires and discussions with the various actors in the field. In particular, the opinions of software developers (practitioners and organizations) were collected and analyzed with special emphasis on the following: levels of qualifications and experience of developers (in both technical and soft skills); the availability, accessibility, attitude towards and extent of use of software methods and processes, together with reasons for same; main performance inhibiting factors and related challenges; characteristics and requirements of emerging software projects; and the interventions required at various levels to improve the existing situation and in the preparation for the upcoming challenges.

Further analysis of the situation, particularly to identify the main factors that need to be addressed by software development methods to improve the existing situation are subjects of the next chapter.

CHAPTER FOUR

4. Further Discussion on the Context and Approaches Explored

In the preceding chapters, detailed discussions and findings were presented on the software development situation in Ethiopia including the challenges faced by practitioners. In this chapter an attempt is made to first summarize the salient features of the existing situation from the survey conducted and the cases analyzed. This is followed by the identification of contextual factors that must be addressed by software development approaches to improve the software development situation (and project success rates).

4.1 Characterization of the Context

No software development process exists in isolation. It derives its meaning from context or it is influenced by the context. Context for our purpose is understood in terms of the environment within which software development projects operate. More specifically, it refers to the types and characteristics of the projects, the profile and skill levels of professionals, the profiles and attitudes of client organizations and users, the profiles and capacities of software development organizations, the level of maturity of the industry and the technological environment, etc.

The following account attempts to summarize the context within which software development projects currently operate in the local setting, based on the experiences and findings reported in previous chapters. The features outlined as characterizing the context, the author argues, have an influence on any software development effort in this environment and as such must be taken into account in all project design and execution activities. In this connection, it is also relevant to note the fact that the context as described may not be considered unvarying over time. It may be repeatable but may never be the same – it will change continuously as a result of the actions/dynamics from within or outside the design and execution of each project. Context is a pattern that exists and moves through time (Bateson, 2000). What is more, change in the context in turn will influence the strategies and actions to be considered in the design and execution of projects. For instance, more project experiences and learning thereof may contribute to improvements in skill levels of practitioners and competence levels of software organizations. As such, context and project actions exist in a co-influencing relationship with one another. For this reason, in a typical project setting, the context needs to be updated continuously by reflecting the effect of project outcomes and experiences. With this understanding, we now proceed to outline the prevailing context at the time of conducting this research. Needless to say, this context is bound to change in the course of time as argued above.

According to the findings of this study, one may characterize the prevailing context as follows based on the findings and experiences documented in the preceding chapters.

<u>The demand: project types</u>

- Most of the projects are outsourced custom-developed applications targeted to address the requirements of specific organizations, at least during the initial development stages. Such application development projects are different from those initiated by software firms for commercial purposes that target the needs of more than one organization as clients.
- Most of the current/emerging projects are very large (by local standards) involving the development of multiple applications packaged together for use by multiple agencies geographically scattered nationwide.
- Most of the projects required business process redesign for each of the applications involved as part of the software development process. Where organizational reform programs were initiated, these were done separately from the software development process.

<u>The supply: capacity of software firms</u>

The software development environment within the software firms is generally immature.

- Most of the private software development firms are small in size and new/young. In most of the cases, they were established, owned and are being run by a group of software professionals.
- The software development processes followed are ad hoc (or chaotic). There are no standards or guidelines defined for processes and methods.
- Due to the absence of historical data, project cost and schedule estimates made are often unrealistic (underestimated) or poorly defined.
- Feedback mechanisms on project progress and processes, and a disciplined approach to project management are not regularly practiced. Information and experience sharing forums are non-existent.

The staff situation within these companies is very testing.

- There is generally a scarcity of experienced professionals.
- There is very high staff turnover.
- There is soft-skill (communication, organizational competence, teamwork, etc.) deficiency among professionals to meet the challenges of the real-life problem environments.
- Most projects are extremely under-resourced. Roles and responsibilities are not clearly defined and communicated. Project staff are overloaded.

<u>User organizations</u>

- Most user organizations are undergoing service reform that involves business process redesign, restructuring, management change, resulting in continuously changing requirements.
- Most user organizations are deploying state-of-the-art technology (hardware) infrastructure.
- There is susceptibility of user organizations to disturbance/turbulence and there is absence of slack resources and plans to manage such disturbance.
- There is very high staff and management turnover.

- In most of the cases, user participation is passive/instrumental and motivation is low (users participate in projects as informants rather than as peers in design). However, there is an increasing recognition among users and software practitioners about the importance of working together collaboratively throughout the various stages of the development process.
- For implemented projects, there are embedments and sustainability related problems.

Support at national level

- There is lack of appropriate educational and training support infrastructure.
- There is absence of home-grown or contextualized methods, as well as absence of national standards or guidelines.

Approaches explored and proposed in subsequent sections and chapters attempt to address aspects of these contextual factors, particularly those aspects that were considered appropriate for handling within the space of this work.

4.2 Solution Framework: Critical Issues to be Tackled

A commonly accepted practice in dealing with the sort of situation at hand is to select and tackle key aspects that will positively impact on the other aspects, thereby resulting in overall improvement of the situation. Adopting this strategy, based on the nature and complexity of the issues involved; in due consideration of the space and time constraints for this research; taking into account the objectives of the research as outlined in Chapter One, in the current work it was decided to focus on the aspect that explores the potentials of methodical approaches to address aspects of the oversize gaps between demand and supply. In due consideration of related discussions in previous chapters, the author argues that efforts to increase the sensitivity of software methods and processes to contextual issues that revolve around devising strategies to bridge the demand-supply gap may bring substantial improvement in the software development situation in Ethiopia.

As indicated, most of the projects are very large (by local standards) involving the development of multiple applications packaged together for use by multiple agencies from geographically scattered locations. Most of the projects have the ambition to change the host organizations by introducing best business practices. The projects also operate in unstable organizational environments. All these factors have exposed the projects to greater risks of failure.

The situation is further aggravated by the lack of capacity on the supply side. That is, the huge and complex software development project demands are in a very marked contrast to the limited and often premature supply capacity of software development organizations operating in the local market. Most of the private software development firms are inexperienced. In most of these organizations, the projects solely rely on few skillful, experienced, and committed professionals. The staff turnover in these organizations is also very high. On the other hand, despite efforts made, foreign software development firms with better capacity do not seem to be attracted to operate in the local environment, for one reason or another (partly to do with economy of scale and local support).

In connection with addressing the supply-demand gap, under the local experience so far, the most commonly proposed advice and strategy, particularly from international consultants, is to scale down expectations and to follow a somewhat 'piecemeal' approach to the introduction of the technology. Such recommendations, according to decision makers and concerned parties, seemed to be very much influenced by the traditional mindset and orientation that promote the thinking that developing countries need to deemphasize investment in the use of technology and instead call attention to the use of other means to address the pressing needs in the area of basic necessity. Proponents of such recommendations were often taken to have the tendency that developing countries talk about the technology and its use often under the influence of NGOs. They were understood to undermine the serious commitment and preparedness as well as genuine needs expressed in some of the countries about the critical role of the technology in the fight against poverty and in the effort to catch up with the rest and maximize benefits from globalization.

The author fully concurs with the viability of project scoping and a step-by-step approach for effective and economical utilization of scarce resources if not with the simple formula of scaling down. On the other hand, it would be amiss for the researcher to disapprove of the 'wishfulness' and 'naivety' connotation implicitly assumed on the part of the project sponsors by some international consultants. This, it is felt, is discouraging at best and derogatory at worst and as such may be counterproductive. From the first-hand experience of working with government-sponsored ICT projects in Ethiopia for instance, the author may take the liberty to acknowledge the level of commitment and determination already expressed in practical/concrete terms by decision makers to realize the ambitious eGovernment programs.

In this connection, it is also relevant to note that the 'scale-down' proposal was not favourably accepted by users in the experience so far (they did not buy into it at all). Despite repeated recommendations made along the 'scale-down' approach, and early failures of some of the initiatives in terms of fully realizing the benefits planned at the outset, users and sponsors have continued to invest more. That is, users do not seem to be discouraged by the initial setbacks and settle for the scale-down strategy on the grounds that such a strategy would not enable them to exploit the potentials of the technology faster to meet the urgent development challenges that they are facing in the other development sectors. In the words of users and sponsors, simply accepting the scale-down solution was tantamount to 'cutting a foot to fit a given smaller size shoe'. Learning from practical project experiences and taking into account the developments in the environment, sponsors and users have demonstrated willingness and flexibility to factor-in lessons learnt in the process of revising plans from one year to the other. At the same time, they continued challenging professionals in the area to come up with better matching strategies to meet the demands.

It is not mainly the purpose of this work to continue such argumentation further beyond emphasizing or establishing the need for understanding this situation in the effort of exploring alternative system development strategies for use in the local setting. There is no point in pursuing the scale-down effort if users are not willing to yield being fully aware of the consequences. In the local realities, we have also seen how overly stretched project time horizons (resulting by default rather than by design) or simply changing suppliers did not help address the situation at all. To this end, the approach adopted in this research is that of tackling the issues from both the demand and the supply sides concurrently. Based on the results obtained from real-life project experiences in the local setting and proven methods elsewhere in the industry, the following specific directions were considered. On the demand side, strategies related to: scoping and prioritizing, rather than simply scaling-down, the ambitions of the projects; using appropriate/contextualized project management and risk mitigation techniques; adopting integrated and collaborative approaches; outsourcing the projects in order to improve the current reality of available competencies in-house; may need to be considered. The project design process must also be based on a consensus view of all main stakeholders. It must be continuously monitored and reconfigured based on progress and changing context. Continuous communication with users and major actors on all aspects of the project (its benefits, consequences and progress) is also essential.

On the supply side, competence development measures at both organizational and individual levels need to be considered. Appropriate strategies are to be devised to establish software processes to improve current realities of organizational competencies. Appropriate learning and training mechanisms are to be devised to improve current reality of individual competencies. For this purpose, strategies that promote contextualized approaches, and that exploit iterative, incremental and collaborative approaches supported by prototyping are to be considered. Most importantly, institutionalization of collaborative approaches between developers and users, based on transformational participation principles need to be addressed. Provisions in the project design to address the skill, time and motivational requirements of realizing collaborative approaches may also have to be made.

Most of the issues and concerns outlined above from both sides are not totally new to software approaches. In fact, there are publicly available methods and processes that cover most of the issues and concerns adequately but for some other context. To this end, for issues and concerns that are already adequately covered elsewhere, the strategy adopted is one of contextualizing them to the local environment. For issues and concerns that are not already covered adequately elsewhere, approaches explored in the course of this research and previous works are introduced.

In order to render the discussion meaningful, the following order of presentation is felt in order. First, background information to be considered in addressing current realities of individual competences among practitioners in the local environment, is presented in the next section. This is followed by a discussion of a case study on the experimentation conducted to explore the feasibility of integrated and collaborative approaches to business process redesign for software development in a real-life project environment. This is presented in Section 4.4. An integrated methodical approach developed for software development and process improvement by contextualizing publicly available methods toward the local realities is presented in Chapter Five. A software process improvement approach based on feedback and collaborative learning to address the current realities of organizational competences among software firms in the local environment is the subject of Chapter Six. Chapter Seven presents additional experience from efforts under way in experimenting with aspects of the approach proposed in this research in two field works: teaching software development in institutions of higher learning, and application software development in Organization B.

4.3 Basic Considerations in Addressing Individual Competencies

4.3.1 Awareness of the Paradigm Shift in Software Development Approaches

The need to shift emphasis from the traditional strong, formal and orthodox approach to systems development to the most recent approaches that adapt toward reality have already been established (Floyd, 1987; Hirschheim and Klein, 1994). The local cases reported earlier also bear some witness to this.

In earlier times of system development, managers were often responsible for providing the system objectives. The systems designer, being an expert in technology, tools and methods of system design, and project management, constructed the system that is said to meet the objectives. Users operate or interact with the system to achieve organizational objectives. In these early approaches, there is an implicit assumption that the ends are agreed. But in reality, ends are controversial and the subject of considerable disagreement and debate. With the perceptions that the pre-specified ends meet the needs of certain system stakeholders at the expense of others, resistance grows and project failure follows. The project with Organization C reported in Chapter Two of this report is one case that bore witness for this in the local context.

In efforts that attempt to address the shortcomings of the earlier approaches, that knowledge about human means and ends is not easily obtained since it is well recognized that reality is exceedingly complex and elusive. The understanding is that there is no single reality, only different perceptions about it (Checkland and Scholes, 1999; Dahlbom and Mathiassen, 1993). Reality is socially constructed and the product of continual social interaction (Pressman, 2003). A better approach is, therefore, to work from within the users' perspective and help users to find their preferred views through continuous engagement and interaction. Through interaction, objectives emerge and become legitimized through continuous modification. In this process, any system that meets with the approval of the affected parties is legitimate.

Recent approaches to systems development advocate mechanisms that facilitate learning by all who are concerned and affected, more than mechanisms based on objective and rigorous methods and tools. Such a paradigm shift not only implies a switch in the role of the developer from one of system expert to facilitator and a communicator who helps to stimulate reflection, cooperation, and experiential learning, but also build knowledge in selected areas of interest. The following paragraphs elaborate on knowledge areas of interest (identified in more recent approaches) that lead to successful software development projects.

4.3.2 Application Domain Knowledge

Needless to say, in any software development, mastery of technical knowledge in software engineering is a pre-requisite. There is now substantial evidence both in the literature (Curtis, 1988) and the local situation reported in the preceding chapters that, in addition to the technical aspect, mastery of application domain knowledge on the part of software engineers is also critical for successful software development. To achieve this, there has to be readiness on the part of software developers to learn about the application domain from their project counterparts in the course of the project design and execution. Traditionally system analysts studied the application system with the purpose of documenting the processes and modeling the revision in the process; the studies were particularly to facilitate: communication with programmers, documentation for future reference, meeting contractual obligation - end-of-phase deliverables, etc. From years of experience in practice, such documents have served best when it comes to meeting administrative aspect contractual obligations and supporting decisions related to selecting general design options. However detailed such reports may be, programmers and users often repeat the process during the actual development of the required software. What is relevant to note in this connection is that, to study a system with the purpose of describing it for others, and to study a system with the purpose of building our knowledge about the domain and then actively and equally participate in its co-construction (business process redesign) collaboratively with domain experts and user counterparts are different. At least, in the latter case one demonstrates the position of a learner rather than that of an expert (which seems to be often the case in the former). The latter requires making additional effort to study the domains that underlie the business process and the interpretation of information obtained rather than simply quoting users or holding them accountable for whatever understanding developers make of the system.

The reception from the user's side also differs depending on which of these positions developers take in the course of the collaboration. In the former case, developers are taken as more of business people whose main interest is business transaction, or fault finders or people that may hold users responsible for whatever they say about the system whenever something goes wrong about the system developed. In the later case, the tendency is to treat developers as co-workers who are there to assist, as people who respect, appreciate and value what users do, and as people who are there to jointly take responsibility in the change they are about to introduce as a result of the redesign. For this and other reasons, users are more interested and motivated to collaborate in the latter setting than in the former. As indicated in earlier chapters, in the cases where the latter (learning orientation) featured, users were found to be less conservative, more open for communication and criticism and learning about their domain and software design.

In view of the foregoing, appropriate learning methods that help software engineers build deep knowledge of the application domain within constraints under which the software development project operates are as important.

4.3.3 Social Skills and Knowledge Interests

To cope with the complexity of today's system development environments, acquiring technical knowledge about engineering and knowledge about an application domain alone are not enough. Any software development process is social as much as it is technical. To start with, the technical software production or construction process has to be managed to be successful. The product developed must be embedded in an organizational environment for effective use. Accordingly, there are lots of people (software engineers, project leaders, domain experts, process experts, user representatives, manager in both client and software organizations, etc.) that work together in the project design, production, embedding and use of the software. These people often work in teams. A team relies on the collective skills of its members because of the scope of the effort, the inherent complexity of the effort, and the number of tasks needed to develop modern software that normally exceeds the ability of any one developer (Sawyer and Guinan, 1998). Team members can feed each other information, stimulate each other in further inquiries, and collectively increase each other's knowledge about both the software and the application domain. For this reason, the social aspect of the software development process, how the teams collaborate and work together in the course of developing and implementing the software, competence in the area of project design and management, and change management, are all as important as the technical aspects.

In similar realms of collaborative work that involves human action and communication, workers (Pressman 2003; Taylor, 2004) identified the concepts of work, mutual understanding, and emancipation as the three fundamental domains around which society and other forms of social organization are arranged. Explaining that systems development is governed by three knowledge interests premised on these domains, Hirschheim and Klein (1994) wrote:

- The technical knowledge interest directs the developer to be sensitive to issues associated with effective and efficient management of the system project.
- The interest in mutual understanding directs the developer to apply the principles of hermeneutics, which examine the rules of language use and other practices by which we improve comprehensibility and mutual understanding, remove misunderstandings, and disagreement or other obstacles to human communication.
- The knowledge interest in emancipation directs the developer to structure systems development to reflect the principles of rational discourse.

Equipped with such knowledge, developers act as emancipators in an attempt to draw together, in open discussion, the various stakeholders involved in system development work. According to Hirschheim and Klein (1994: 1208), to succeed in such communicative action, developers need to take note of the following typical obstacles to human communication throughout systems development:

- Authority and illegitimate power – these create anxieties and cause people to distort or withhold information in order to protect themselves.
- Peer opinion pressure ("group think") - it creates tunnel vision for the sake of loyalty, reducing the validity of judgments by suppressing possible validity checks through criticism.
- Time, space, and resource limitations – these prevent universal access to knowledge even though in principle it is available. This includes the common situation that knowledgeable people remain silent due to lack of motivation to participate because of work overload or the socially created need to withhold important information unless it is to one's advantage to engage in a debate.
- Social differentiation - differences in the level of education, specialization and personal values and beliefs increase the risk of misunderstanding.

- The bias and limitation of language use – these distort perceptions and lead to narrow problem definitions through jargon and cognitive anchoring.

In a related work, Diwan et al. (2002) identified three skills necessary for successful group work:

(i) an appreciation for interdependence and the ability to recognize when a task is dependent on the efforts and accomplishments of others;

(ii) the ability to consider and argue for or against different viewpoints in a constructive manner (negotiation); and

(iii) familiarity with structured, systematic decision-making procedures (group problem solving).

Other workers (Tan, 1994) emphasized effective communication skills as success factors. Securing effective collaboration and cooperation with users from the early on requires consistent and effective communication of goals and objectives as well as roles and benefits, at both organizational and project levels. According to Tan, such effective communication between users and developers may be achieved through establishing rapport, shifting perspectives and effective management of communication transactions. Effective communication often leads to mutual understanding (Tan, 1994), mutual respect, closeness and shared purposes. Where such mutual understanding is absent or not intentionally introduced early in the project design process, at best collaborations seemed to struggle or suffer throughout the development, installation and use. At worst, as evidenced for instance in the case of Organization C reported in Chapter Two, this may lead to conflicts that cause resentment and hostility with the potential risk of aborting the project and ending up in litigation (which may not do any good for all parties involved). In view of this and related discussions and experiences reported elsewhere in this report, we believe or strongly argue for effective communication as one of the critical factors both in motivating users to actively participate in the development process and in achieving a successful software project in the local context. Accordingly, this must be intentionally and skillfully considered in the project design phase and continuously maintained through out the project period. From the author's experience in the local setting, establishing rapport with users helps the most in this connection. This may require getting closer to the user and spending some time with the users at their premises – to create the feeling of ‘being there for’ and ‘being there with’ the users.

During the discussions held as part of this study, both developers and users who were actively involved in the testing and finalization of installed systems, reported that they have practically witnessed that the number of enthusiastic users increased (in some of the cases dramatically) once they were able to establish person-to-person (peer level) relationship bypassing the formal business relationship that the project contract dictates. According to individual participants, most formal contracts signed between suppliers and clients basically lacked jointly developed components that address effective communication during project execution. Although considered valid administratively, the contracts were often criticized by people in the lower ranks of the project for introducing unnecessary biases and complication (defensiveness, control-orientation, etc.) into the communication between users and developers.

4.3.4 Methods Adapted Toward Reality

Among the mechanisms incorporated in contemporary software methods and processes to address the various challenges and related uncertainties in real-life software development are: iterative, prototyping and collaborative approaches. Each of these is briefly explained in the following paragraphs.

In order to mitigate the risk of getting the requirements wrong at the beginning, developers and users collaboratively iterate around the requirements. The iterative software development process is better described by the argument (tending gardens metaphor) forwarded by Hunt and Thomas (1999), that software is more like gardening than it is like construction. As explained by Hunt and Thomas, software is

> “… more organic than concrete. You plant many things in a garden according to an initial plan and conditions. Some thrive; others are destined to end up as compost. You may move planting relative to each other to take advantage of the interplay of light and shadow, wind and rain. Overgrown plants get split or

pruned, and colours that clash may get moved to more aesthetically pleasing locations. You pull weeds, and you fertilize plantings that are in need of some extra help. You constantly monitor the health of the garden, and make adjustments (to the soil, the plants, the layout) as needed."

The iterative development process is usually supported by a prototyping mechanism. Prototyping facilitates focused discussion and agreement with stakeholders in terms of clarifying requirements. A step at a time (incrementally) in the development process, the prototype evolves into a fully operational system. According to Hall and Fernandez-Ramil (2007), prototype development offers:

- interaction with people and their problems;
- a reduction in uncertainty concerning the usefulness of the software; and
- a short delay between producing code and seeing something working.

A common practice in prototyping and incremental processes, apart from the iteration, is collaboration - the involvement of users and stakeholders throughout the development process. In such collaborative approaches, users and their representatives are involved throughout the design and development process by joining the development team which adds to the team an essential usage perspective. Various types of user participation are discussed in literature (Hall and Fernandez-Ramil, 2007), ranging from: *consultative* participation where representatives of stakeholders reach agreement, *representative* participation where members from all groups involved in the change are able to influence the nature of the new systems, and *consensus* participation where everybody associated with the business processes involved is able to play a part in the design of the new system. Participation can also be distinguished as *transformational* participation, where participation is an end in itself, and *instrumental* participation, where participation is a means to an end (Dahms and Faust-Ramos, 2002).

Beyond ensuring that the functionalities of the system to be created and introduced are socially appropriate user participation is also important to realize user-centered designs that ensure the usability of the system at the level of interaction between the computer and its user (Hall and Fernandez-Ramil, 2007). For this reason, while contemporary methods for information system development generally accept that users should be involved in some way (Jepsen et al., 1998), the form of the involvement differs considerably. Mostly, users are viewed as relatively passive sources of information, and the involvement is regarded as "functional," in the sense that their participation should yield better system requirements and increased acceptance by users (Clement and Van den Besselaar, 1993). For instance, this was partly the problem at Organization C discussed in Chapter Two – the participation was more ‘passive’ and ‘functional’ rather than transformational. Although users were said to involve in the project design and implementation, scratching beneath the surface, the project was not fully participative. There was no meaningful user participation in all phases of the design process. The participations were more for external than for internal consumption – apparently, the participations were limited to membership in a committee and attendance of meetings, rather than influencing the process and outcomes of participation in the software development and its implementation. It was also the case that even the user representatives and members of the committee already had heavy existing workloads and did not have enough time to invest in the project or were not productive in the process as a result of workload in their work places. Access to relevant information was restricted to technical project staff and management only. The skills, experiences, creativity as well as knowledge of domain experts were not utilized in the customization processes.

According to Dahms and Faust-Ramos (2002:281), true user participation changes the design and development process into an evolutionary process of mutual learning and co-operation between designers and users about technical possibilities and useful development of these possibilities. They identified three reasons for involving future uses in the system development process:

- It may improve the knowledge upon which the system is built and therefore make it fit better to the given context;
- It may enable the users to develop realistic expectations of the system and may reduce their resistance to change;

- It may increase the local democracy by giving future users the right to participate in decisions that are likely to affect them.

According to Robey and Farrow (1982), among the benefits of user participation are:

- more accurate assessment of user information requirements,
- prevention of costly system features that are unacceptable to users,
- greater user acceptance and support of the system,
- improved user understanding of the system, and
- granting of democratic rights to organization members.

The benefits from such participative and collaborative approaches are obvious. Such benefits are even too evident in the local case, particularly with custom-developed business application software. Custom development involves people and activities at both software developer and client organizations. It requires and involves more interaction, convincing and consensus building within the client organization, much more than learning about application domain knowledge, requirements and design preferences to develop the product. The main issue and concern is rather how to implement or realize such participative and collaborative approaches in software development projects.

In summary, as indicated in the forgoing, to cope with the complexity of today's system development environments, acquiring technical knowledge on engineering and application domain alone are not enough. In fact, most of the problems of software development, as documented in the preceding chapters of this report, are rooted in the non-technical aspects of the software development process. As such, systems developers have to be ready to address knowledge interests in the non-technical areas discussed above (i.e., effective communication, experiential learning and cooperation. Concepts, techniques and tools that address such non-technical aspects need to be incorporated in software methods and approaches. As indicated, the more traditional approaches have totally neglected this dimension. The most recent approaches recognize and advocate the importance of work in this dimension. However, still approaches that provide techniques and tools to effectively address these issues (both in practice and teaching) are yet to be developed.

In this research, attempts were made to explore ways and means of realizing collaborative approaches in the process of software development. An aspect of an extensive experiment done in this direction over the last four years, particularly in connection to business process redesign for software development, is presented in the next section. Related aspects under investigation (by drawing lessons from this experiment) and proposed for consideration in the methodical approach proposed in this research, are presented in subsequent chapters of this report.

4.4 A Collaborative Approach to Business Process Redesign

Although business process redesign is not an activity often explicitly specified as part of the mainstream software development activity, it featured manifestly and consistently in the local situation. Hence efforts were made to explore the possibility of customizing methods suggested for business process reengineering and systems analysis/design for the purpose of applying same to the redesigning of processes for software development. Before reporting on the efforts made in this direction, an attempt is first made to establish the need for it.

4.4.1 Why Business Process Redesign

All organizations do work, undertaking coherent sequences of activities in order to achieve the objectives of the organization. The sequences of activities that constitute the work are usually referred to as business processes (Hall and Fernandez-Ramil, 2007). Business application software is developed with the intention of supporting these business processes. Such is the association and interdependence between business processes and application software. Accordingly, any change in one will have an implication on the other.

Nowadays, it is customary for any modern organization to redesign its business processes to devise new ways and means of doing business or providing services or to stay competitive, or to improve productivity and quality, or to satisfy customers, or to cut down costs, etc. Such redesign activity may be carried out as part of an organizational reform initiative or through the implementation of software systems that introduce best practices. Where the software development project comes after the reform or as part of it, the redesigned business process forms the foundation for software design. On the other hand, where the introduction of software is used as an instrument for the business redesign, one of two options may be considered. If the software is to be custom-developed, the business process redesign may be performed as a front-end process to software development. However, if the acquisition strategy considered is to purchase industry standard off-the-shelf software, the process redesign may come at the end, as part of the organizational embedding of the software. It seems, therefore, one way or the other, business process redesign is unavoidable either as a front-end process or as part of the smooth implementation and use of the software for organizational purposes.

According to Jones (1997), redesign of business processes must be conceived as the essential first phase of any substantial development effort.

> "Not every process must be redesigned radically, as in reengineering, but every process touched by automation or new work requirements must be considered fair game for redesign …. Failing to take a strategic business perspective and missing the rare opportunity to create a better process will diminish competitiveness and growth" (Jones, 1997:225).

As such, the first area of concern in any opportunity for new system development is to review the underlying process that the business relies upon. This is very essential particularly in the local situation where the software development projects are considered to be part of a larger organization-wide reform movement that aims at introducing a whole new way of doing business that transforms a control-orientated service process to customer-orientated support process.

Taken together, the type of business processes involved in most of the organizations in Ethiopia are a mix of standardized and well-structured routine processes (such general business applications as general accounting, payroll, stock control, etc.) and unstructured decision processes (such as granting permits, deciding on discounts, assessing claims, etc.). While the former represent processes where mass-production of a type to process many cases at once are possible, most of the latter involve processes in which individual cases are dealt with in more or less direct contact with stakeholders. In the latter cases, at times, the officials or operatives in charge of the processes may need to involve additional actors - asking a colleague for help or organizing a meeting to jointly decide on how the transaction should be handled. The outcome of such consultation may alter the sequence of work processes. It may also need the invention of workarounds to handle specific cases.

Accordingly, in the effort to develop software based on existing business process where there are more workarounds than the properly documented and publicly available business processes, one has no choice but to rely heavily on the existing workforce who often (for one reason or the other) are disinclined (or at best undecided) to change existing practices or introduce best practices that demand transformation of existing practices.

The situation is further complicated when one deals with applications whose processes cut across multiple organizations as typified by most of the eGovernment projects. In the design and implementation of software solutions for such cases, adequate consideration must also be given to the often non-trivial characteristics and requirements of inter-organizational service processes. Take, for instance, the case of customs, banking and transport agencies involved in the processing of a certain import transaction (a transaction that involves organizations in more than one sector); or the case of a clinic, a hospital involved in processing a certain patient record (a transaction that involves different organizations within the same sector); or the case of a ‘sub-city’ and a ‘municipality office’ involved in processing a certain building permit processing. In the course of processing such cases, various documents have to be exchanged, some of which are delivered by the subject while others are sent by messengers, mail or fax or other means. There are information integration (centered on facilitating information flow) and process integration (centered on interrelating steps and stages of process performance) issues across technical and/or organizational borders (between different agencies and within the same agency at local, regional and federal levels) to be considered as factors that influence the ICT solution design and implementation (Klischewski, 2004). That is, appropriate standardizations, interoperability, flexibility and customer orientation need to be considered in the design of software solutions for such applications. Per the requirements of the reform initiated in the public service organizations, the processes need to be: transparent to the parties involved, customer-centred and result-oriented. To support this, the underlying information systems (database, software, hardware, etc.) need to be able to communicate with each other. The exchange of the information (information items and data elements) needs to be standardized to provide flexibility in process execution. All these cannot be addressed without a serious business process redesign undertaking.

What is more, in the local setting, most of the tenders floated for software systems development require business process redesign as one major activity to be carried out, in fact before the actual development of the corresponding software. For this reason, this activity becomes one of the major requirements to be explicitly addressed by software processes and methods to be used in the local setting.

Many methods may exist in order to deal with business process redesign issues, ranging from the most radical system overhauling to continuous incremental improvements. What follows is a generic approach that evolved out of the experiments carried out with real-life projects first at Addis Ababa University AAU and then at Organization C of the cases reported in Chapter Two, over the last four years.

4.4.2 First Attempt at AAU

The method applied in this experiment mainly originated from the works of the author in teaching and practicing systems analysis and design at AAU and the cases reported in Chapter Two respectively. Additional motivation for the collaborative approach incorporated came from the author's active participation in the self-assessment and peer-review exercises of the comprehensive university-wide program review project at (AAU) during 1997-1998.

The work took place in 2001 at Addis Ababa University (AAU) when the author was given the responsibility to lead a project that aimed at redesigning of the various business processes of AAU as part of the preparation for full scale automation. Before the year 2001, a number of automation projects were initiated by AAU with a view to improve service provision and support. Among the major areas covered in these initiatives were: library service management, student record management, and integrated financial and administrative support. Unfortunately, most of the initiatives experienced difficulty in terms of meeting expectations. The projects for the most part simply automated existing activities – merely changed the manual operations to computer-based operations. The information was the same, the process was the same, etc. According to the top management of AAU, this however was not sufficient and did not help in overcoming the limitations of the manual systems. After an assessment of the situation, the AAU management decided to commission a new project in 2001 to develop an integrated business application software system based on best practices in the sector. The project involved business process redesign as a front-end activity.

In preparation for the work, a critical assessment of the previous experiences both at AAU and projects in which the author was involved elsewhere were made. Based on the lessons drawn from these experiences and in due consideration of the scope of the work and the variety of business areas, a collaborative approach was considered appropriate. The environment at AAU at that time was also supportive to such a system of work. Accordingly, collaborative teams composed of members from the academia, operatives and senior students were established in respect of each application area. For instance, in the case of the finance application area, a group was formed composed of staff members from the Accounting Department of the Faculty of Business and Economics at AAU, the Finance Department under the Business and Development Vice President, and the Faculty of Informatics. Wherever possible, particularly in case of operatives, the unit heads were selected to join the team. Members from the academic and operational units were to seriously involve themselves in the re-design of the accounting functions and processes (by bringing together knowledge and experience from both the work area and from the academic field and best practices elsewhere). The contribution from the members of the Faculty of Informatics, according to the original plan, was mainly limited to the design of software to support the redesigned business process. As required, these members were to be supported by senior students from the Faculty who are to join the team later at the time of programming. Similar arrangements were made for other areas of business for the teams to work in parallel. The other business application areas included were: procurement, human resource management, library management, registrar (student record management), and facilities engineering and maintenance.

To coordinate the activities across the groups, a steering committee was established. The steering committee was composed of group coordinators and representatives from the central management. It was chaired by one of the former presidents of AAU in due consideration of his familiarity with the system and problems. The researcher participated in the project in two roles: as a sponsor (in his capacity as a member of the AAU management) and as a project leader where he actively but informally participated in the technical matters related to process analysis and design.

Actual project activities were carried out by individuals and pairs of individuals through interviews and document reviews. While the operatives were charged with the task of reviewing existing practices within their respective domains, their faculty counterparts were responsible to bring in best practices from their previous exposure, experiences and the literature at large. Each group had a weekly meeting where it exchanged information, discussed design options and reviewed the progress based on reports from individual and pair assignments. Based on the individual and group level discussions and reviews, each group tried to document in detail the existing process together with its limitations and a proposal for its improvement, in its respective area of assignment.

Soon after each project took off, certain problems were reported. The motivation of the team members from the operation side was low. In the discussions conducted to investigate the situation, the people from operations complained of workloads in the normal (non-project) routine jobs. Further probing into the situation also revealed that they had complaints on what they called ‘double standards’ in the treatment of team members when it comes to recognizing their contributions. In particular, per the arrangements made at the beginning, time spent on the project by members from the academic side was compensated. This was not the case for the operations. The assumption was that for the people from the operations group, this work was part of their regular duties, while for the others it was not and therefore needed to be compensated.

To expedite the project progress, corrective measures were taken in terms of establishing incentives to the people from the operation. In addition, instead of relieving them fully from their normal routines and responsibilities, an understanding was reached on how much time from office hours they should spend on the project. The same was communicated to all concerned including their respective supervisors. This intervention helped for the project to run smoothly afterwards.

Each group reported its work both orally and in writing to the steering committee on a weekly basis. Regular reviews and discussions were held at the steering committee level on monthly basis. As required the steering committee also met weekly or fortnightly. The steering committee also served as a critic for all group works. For this purpose, in addition to circulating relevant documents ahead of time to the members of the steering committee, each group made presentations at the meeting about its findings and project progress. This was followed by a series of questions and challenges by other members. Each meeting was concluded with suggestions to enrich the work and discussion on updating plans based on the outcomes of the discussions. Problems encountered during the course of the work were reported together with the efforts made to tackle them. All outstanding problem referred to the steering committee were discussed and resolved. Problems that were not resolved at the steering committee level were referred to management and the chair person of the steering committee usually took up the matter with the management. Accordingly, based on the reports circulated and presentations made, detailed deliberations were made to streamline the processes, resolve conflicting proposals across processes, and address problems encountered during the project process.

> To learn about the domain and best practices, team members used mainly interviews, face-to-face discussions, and review of practices elsewhere. The group used for the most part free-format based natural language text, work-flow diagrams, such project management tools as network charts (PERT) for process activity charting and sequencing and the usual meeting protocols. Extensive informal consultations were made among experts in the various units.

The group submitted its report within six months. According to the project evaluation by the group itself and the AAU management, the project success was rated high at least for the process documentation part. As a result a go ahead decision was obtained from the AAU management to proceed with the development of the required system and software to implement the recommendations. Unfortunately, the author had to leave AAU at this stage. Based on the information obtained through colleagues, the project continued as part of the reform program of AAU but not with the same speed and momentum as before.

4.4.3 Second Attempt at Organization C

The author was charged with a similar project at Organization C (of the case reported in Chapter Two) in 2003 but at a much larger scale. The assignment brief was to completely overhaul Organization C through IT. The engagement at Organization C was huge and extensive. It went on for more than two years. It involved multiple stakeholders ranging from the labour force to the board of management and supervising authorities (including Ministers). The work reported below concerns an aspect of the undertaking that relates to business process redesign as part of the software development project for administrative and customer support services.

From the outset, a decision was made to build on the experience at AAU. As part of the initiation process, an assessment was done first to draw lessons from the AAU experience. The assessment was made through a series of reflection sessions with selected members of the AAU project (particularly with those members identified for engagement at Organization C as external experts in their respective professional domains). The assessment focused on identifying areas that needed further adjustment in the approach employed for the work. Among the major points that came out from the exercise were the following shortcomings in the earlier effort.

- The process redesign at AAU was mostly confined within functional units. For this reason, the levels of integration between processes within the different functional units were not explicitly and adequately addressed. For instance, with regard to financial management, differences in viewpoints between the finance unit and the human resources on matters related to the payroll process; differences in viewpoints between donors and in-house experts on addressing donor requirements related to the processing of donor supplied funds; issues related to reconciliation of bank statements with in-house records; etc., that were encountered in the process of consultations made for the work were not exhaustively addressed and worked out between the units. Such issues that cut-across functional units were included in the redesigned system in their original form.

  The separation of the business process redesign activity from the actual software design process, particularly the arrangements made for members of the IT (software) group to join the business redesign team at a later stage in the process was inappropriate. Not only did this contribute to the delay in the taking-up of the project but it deprived the software experts of the opportunity to learn about the business process both existing and proposed. (Originally, the decision for the IT group to join the team at a later stage was made mainly for the purpose of avoiding tendency of IT experts to jump to devising technical solutions for all types of problems.)

- The level of participation from the operations side was limited to experts assigned to the team and those consulted in the process. The viewpoints and experiences of the people directly involved in the day-to-day process were not adequately considered. Workarounds encountered in the day-to-day operation were not adequately dealt with and were left for consideration during the actual software developments. What is more, even the experts who were on the redesign team from the operation side took the assignment as an individual engagement and contribution. As such, they did not make any effort to try out or introduce aspects of the design (what they have learnt in the process) in their respective operational areas. There was a lack of ownership and follow-up.

Based on these observations, reflective workshops were held directly with workers in selected units (for instance, customer services from front office, and procurement from the back office). The discussions focused on interactive processes in which participants are engaged in a process of identifying key issues. These were made based on stories told by participants to expose such problems as: complaints filed by customers on services; duplication of activities and functions; waste or abuse of resources; delays in decisions and their consequences; corrupt practices; abuse of authorities; etc. This was followed by analysis of the stories told and the issues exposed. This was necessary in order to filter out and structure relevant issues that were considered critical to understand more about the situation at hand and to integrate insights into the project design and for the purpose of customizing the earlier approach used.

Among further adjustments introduced to the earlier method based on the feedback and analysis made, were the following.

- To introduce three working groups to carry out the activity: operations group, critique panel and IT group (see below for details).
- To involve users as much as possible in the process to benefit from their experiences.
- To use workshops as the main vehicle of requirements definition instead of interviews and one-to-one discussion sessions.
- To align the project with corporate reform programs.
- To use awareness creation workshops throughout to continuously inform stakeholders and the community on the purpose and intent of the undertaking, on roles and responsibilities, on the progress of the project, etc.

- To devise and use simple documentation and communication tools drawing on project management and Joint Application Development (JAD) techniques and tools.

Per plan, each individual worker in the business areas addressed was required to document his or her routine activities – activities that are performed in the course of discharging their duties and responsibilities. For this purpose, in addition to the discussions in a series of workshops on the purpose and goal of the reform and the benefits thereof, the workers were given orientation on the techniques and tools to be used for the documentation. To create awareness among the critical mass, the corporate wide forums created to support the national reform programs[8] coordinated by the Ministry of Capacity Building were used.

The operatives, the people already working in the functional areas, were then guided to document their existing functions by simply listing the tasks carried out in respect of each process using a simple format adapted from project management techniques. Each task was listed in terms of: a task number used as a reference and identification, a brief activity description (not more than one statement), duration (time required to perform the activity in terms of minutes), responsible unit, dependency (by identifying predecessors for the task described in terms of task numbers). The business process mapping was then generated from this documentation, using project management tools (network chart). In the process, external consultants that specialized in the business area of Organization C were employed to provide additional technical assistance to the in-house group. Figure 1 and Figure 2 below show examples of an Activity List and a Process Mapping respectively.

[8] As part of the national transformation program, a series of discussions were held to introduce the various national strategies in general and the reform programs in the areas of strategic planning process, business process reengineering and result-based employee performance evaluation in particular. Every employee is expected to attend.

Activity #	Activity Description	Time	Responsible unit	Dependant
1	Project proposal	3.7d	AND	
2	W.O	10d	CPBD	1
3	Asphalt permission	18d	AND/Munic	2
4	Material request	4d	AND/LD	2
5	Budget transfer	10d	CFD	2
6	Transport arrangement	10d	LD	2
7	Transport material	20d	AND	3,4,5,6
8	Assign/Dispatch team	2d	AND	3,4,5,6
9	Trenching	15d	AND	7,8
10	PVC laying & back filing	4d	AND	9
11	Manhole/Pit const	52.75d	AND	7,8
12	CC foundation	21.5d	AND	7,8
13	Duct PC installation	7d	AND	10,11
14	Direct buried PC inst.	10d	AND	9
15	Sec under gr. cab inst	33d	AND	9
16	Over head cab inst	21d	AND	7,8
17	Jointing	84d	AND	13,14,15,16
18	Testing/Handover	30d	AND/TBS	17
19	Return	2d	AND	18
20	Closing W.O	3d	AND	19

Figure 1: AN Construction Activity List

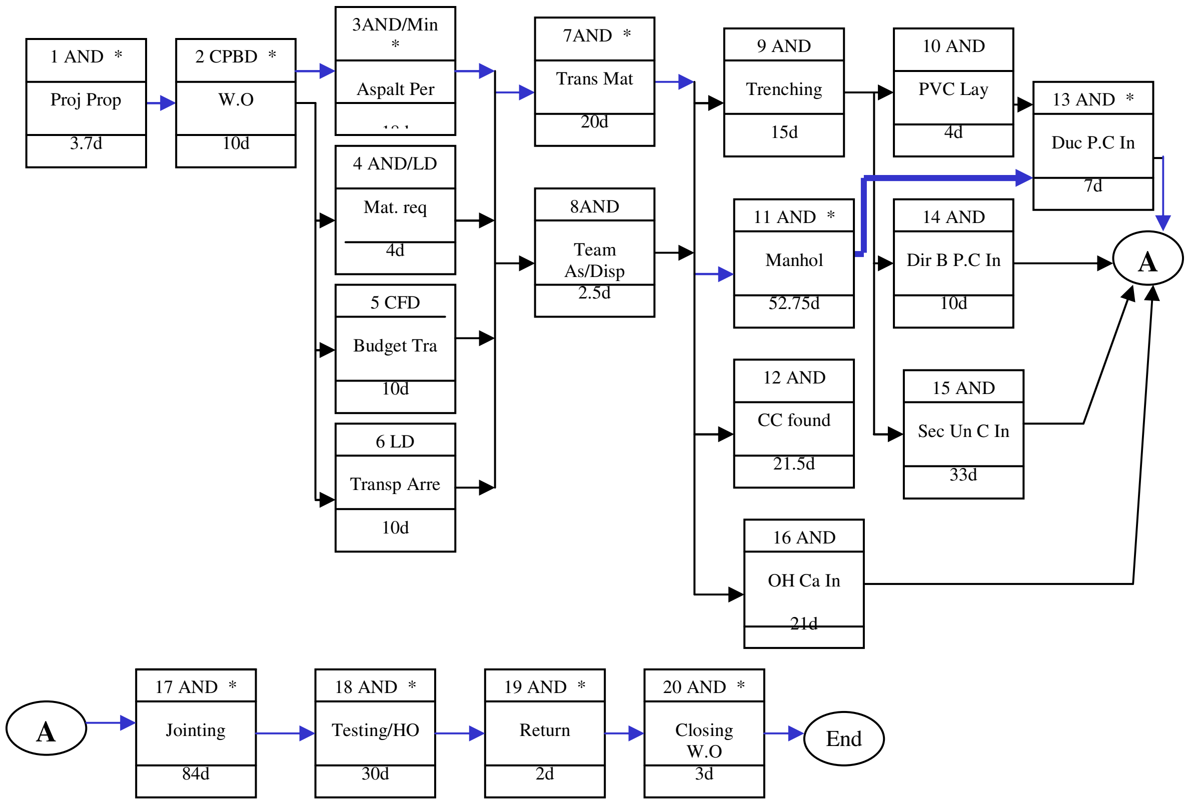


Figure 2: AN Construction Process Mapping

The documentation of the process for the most part was based on what was actually being done in reality instead of referring to what aught to be as stated in the procedure manuals. This gave an opportunity to capture the various workarounds introduced to handle special transactions that the existing procedure manual failed to address or failed to provide guidance for. What is important to note here is, the departure from the traditional approach where the analyst talks to the users on what and how they do their work and then document same for use in the subsequent activity of software design. Instead, the people doing the job (users) themselves were made to document the processes with a little guidance. This, as can be noted, is also a further step from the approach employed in the second round development project of both Organization B and the AAU project.

In actual practice, the list of tasks was developed as follows. First, each individual person was made to document what s/he did using the format but without bothering to complete the details. Then those working in the same section came together to review and synchronize their lists; similarly sections worked together to review and synchronize the lists; and so up the hierarchy. As the size of the group coming together grew, to have a manageable size and to render meaningful discussion and review, lower level work units were represented by a group delegated for that purpose. From this we formed what we earlier referred to as 'Operation Group'. As required, the operation group had taken two or three days away from the office to discuss and finalize the documentation and discussion. The system so documented by the operatives themselves (the Operation Group) was called the 'As-Is' system. The Operation Group was then tasked to develop, in consultation with the employees involved in the process and the stakeholders, and based on series of envisioning exercises, a proposal for revising the same business processes. This proposal was called the 'Should-Be' system.

> What is worth noting in connection to the process documentation is that initial attempts were still confined to activities performed within the functional units. It was through a serious of joint reviews and deliberations that these were extended across functional units, to include all those involved from the start of the activity to its end. This was another major lesson learned. Particularly, initially the different functional units involved were not aware of the implications of their task on others and only cared to address their part without bothering to what extent this may or may not have contributed to the overall accomplishment of the main process. Users, particularly those that participated in the workshops where these issues were demonstrated and discussed, were taken by surprise when they came to know the cumulative effects of their activities. In particular they were surprised to learn the huge amount of control-oriented activities and paper work involved, complications in communication (back and forth routing of papers) and related duplications of efforts (tasks common across the offices), processes that generate work and outputs that no one truly needed any longer, and preparation and distribution of unnecessary paper copies, unfriendly forms and formats, etc. and the huge amount of time spent in the process. This became the source of appreciation and inspiration that led them to actively and continuously participate

in the efforts. Earlier workshops among various functional units that used to be dominated by pointing fingers, externalizing shortcomings, putting others to blame, etc. changed after this observation to forms for conducting lively and constructive discussions and experience sharing.

The As-Is system and the revisions proposed by the operatives were then subjected to review in a Critique Panel.

Rationale for the Critique Panel: it was believed that depending more on users for the ‘Should-Be’ will limit the chance of introducing innovative solutions. As much as it was important to understand the problems that users currently faced and their expressed requirements for improvements, attempts should also be made to anticipate future needs, and invent a system to both help users solve their problems and provide for the anticipated future needs. For this purpose, we introduced the concept of Critique Panel – which involves engaging stakeholders outside the Operation Group in the vision creation process[9]. Members of the panel were composed of individuals outside the operation group (see below for more).

This was also in a contrast with the notion of use context or nature of agile methods that heavily depend upon the ‘customer’ to create a vision of the software under development. This may work under circumstances where customers are skilled and up-to-date with best practices in their area of work and profession. This, unfortunately, is not the case in most developing countries in general and Ethiopia in particular. Under the circumstances, therefore, depending on those staff members/operatives currently involved in the operation may not bring the required level of improvement or can only bring very limited improvement. It may also be considered as placing a large and sometimes unrealistic burden and expectation on the users who have not had any chance of upgrading their skill for some time. In this connection, it is relevant to note that, in the local setting, we are talking about workers that have been practicing the

[9] This panel concept may be considered as a substitution for the instructor or coach in the original proposal of Schoen’s reflection in action (discussed in the next chapter).

same routines for years, and who did not bother or take initiatives, for one reason or another (lack of capacity, lack of authority, lack of concern, lack of motivation and incentive, etc.), to fix system problems. When process related problems occur during operation, the tendency is to refer the matters to others or temporarily mend them or invent workarounds to manage incidences on a case by case basis. In the existing situation, particularly in government and public service organizations, most employees are not motivated enough to venture into doing their job in an efficient manner, let alone envisioning the future process. To make matters worse, there are often negative forces operating with vested interest but under the guise of concern/control.

Accordingly, for each Operation Group a corresponding Critique Panel was organized and commissioned. The panel was composed of critics from various units from within and outside the organization but with a good deal of knowledge on the domain of discourse (the business process under reference) and preferably with some prior exposure to best practices in the industry. The overall purpose was for the Operation Group to present and defend their proposal in front of the panel, and for the panelists to challenge the operatives through constructive criticism based on customer needs, expectations and current industry/best practices. Beyond questioning, they also provided 'why not this way?' type of proposals for consideration by the Operation Group. The end game was for the group to critically examine the existing ways of doing things and suggest modifications, and brainstorm on alternative options. Through joint discussion and negotiation, both groups had to agree on the ways in which work should be redesigned and done in the future (that is, revise the earlier version of the 'Should-Be' system). As such the panellists are used for passive feedback and critic. They are used as a source of insight and detached reflection, unbiased, uninvolved perceptions, and to bring the 'voice of stakeholders' into the redesign process. For this to yield better results, based on repeated trials during the piloting, a discussion modus operandi was agreed upon, which stated among others to: involve all members of the group, foster creativity, and respect all ideas. As required, in some of the cases, representatives of customers were also invited to open meetings to express their expectations and to comment on the process redesign plan under consideration. The experience from such exercise indicated the importance/benefits of including visionary customers in the Critique Panel.

Then the third group, the IT Group that mainly was composed of software engineers, was given the chance to check the extent to which the suggested changes have taken advantage of the new opportunities provided by the technology. The focus here is one of fitting technology into the ‘Should-Be’ system to support the newly designed business processes and work practices. As a matter of principle and strategy, it was agreed not to miss out on the new opportunities provided by new technology in the design process. For instance, with regard to the project under consideration, in line with the overall plan to introduce customer-oriented one-stop shopping services, the group was expected to explore possibilities of using the Internet, the web or hand-held wireless accessories to extend the one-stop shopping to no-stop shopping model. In particular, this referred to a fully electronic environment so that customers with necessary resources would access the services from wherever they are without necessarily physically approaching the service provision stations.

Finally, a team composed of representatives from each of the three groups was organized to sift, analyze and combine the various ideas discussed and proposed at the various levels and stages to yield one or more possible solutions for consideration by management. As required, simple prototypes were developed to demonstrate aspects of the solutions proposed (for instance, the human resource business process redesign was one such case). To do this task effectively and within short period of time, sometimes arrangements were made for the team to go away from the office (fully equipped with necessary resources) for a couple of days. This was made on the belief that the team would function better in an environment away from everyday pressures. With such arrangements, participants focused better on the process and reflected on the discussions and suggestions made at all levels. Upon return, the results were presented to senior management and members of the Operation, Critique Panel and IT groups in a series of de-briefing sessions. The management then decided on one option and took necessary steps for the implementation together with the concerned units.

In the case of Organization C, partly influenced by the organization-wide movement to overhaul business processes, some of the recommendations required questioning the policy premises and mandates. Where this was the case, another round of discussions and reflections were made with policy re-design in mind. There is no space here to report on the details at this level, nor is this necessary also for reasons of confidentiality. Nevertheless, discussions at this level necessitated the involvement of other stakeholders within and outside of the organization, including the board of management, the supervisory authority and other public and private agencies. Most of the policy level revisions were taken up at the board of management level. For this purpose, the management, the labour union leaders, and other concerned parties met on weekly basis to discuss the issues, brainstorm on options and come up with recommendations. Interestingly, what is relevant to note in this connection is that this level consideration provided an opportunity to experience the double-loop learning discussed in Chapter Six. In the discussions at this level, both the approach used in redesigning business processes and the recommendations in some of the cases (particularly in areas that affected large members of the labour force such as maintenance and collection services) were seriously challenged and questioned. Serious arguments were raised on the extent to which the proposals for the 'Should-Be' system would address problems identified with the existing system. Critiques were forwarded on the overemphasis on technical and infrastructural issues and less emphasis on the human dimension. Decisions were made to revisit the recommendations in the light of work culture, attitudes and competence of the workforce.

Emphasizing the implications on methods, the deliberations at this level partly helped to refine the approach used for the business process redesign. In particular, three more dimensions were incorporated in the business redesign approach:

- To include labour union representatives in the various working groups;
- To introduce a support infrastructure to address organizational communication issues related to the activities of business process redesign, particularly to inform the community about the redesign activities, solicit suggestion and comments on related issues and required changes that are expected in the work culture, skill sets and corporate values in the new environment; and

- To increase emphasis during redesign on integration of functions around processes and human resource development issues.

Subsequent workshop sessions focused on resolving the issues identified at various levels and discussing implementation related issues.

In parallel, the in-house software group continued the development of the prototypes that captured the revised business plans. The prototypes were further revised in close consultation with and involvement of the Operation Group and converted into mature systems at least in some of the cases (human resource and finance subsystems). A decision was then made to operationalize the systems developed by making related revisions in the work processes and providing additional training for users. At about this stage, the author discontinued day-to-day engagement with the project due to some other assignment.

During the final stages, people actively involved in the process redesign were able to reflect on the overall impact of the approach. On the whole, the results obtained from the experiences were considered encouraging. The collaborations of actors through the three groups did provide suitable environment and platform for learning and collaboration. Lessons learned from further analysis of the experiences were used to further refine the method and integrate it into prototyping-based software development. An attempt is being made to explore the application of such an approach in the project at Organization B and an upcoming e-government software development project at AAU. An aspect of this has also been incorporated in the approach proposed in Chapter Five.

As a process documentation technique, in addition to Activity List and Process Mapping presented above, one could also use Cooperation Picture of Wetzel (2001) to describe the different set of tasks and activities from the workplace perspectives. Cooperation Picture relates to representing the work objects and information to be exchanged between all participating parties or ‘actors’. Further still, storyboarding techniques suggested in CD could also be used to capture details of specific tasks in the processes (work practices).

The next two chapters will introduce a methodical approach considered appropriate for the local setting. The approaches proposed are customized versions of popular methods in the industry. The customization mainly involved incorporation of valuable lessons learned from years of practical experiences (in both teaching and professional practice), and modifications of aspects of the original methods based on the inputs from the findings reported in the preceding chapters.

CHAPTER FIVE

5. Reflective Steps: the Proposed Approach

"No life is as simple a progression as an academic vita outlines .. We go on, and on, and on, arriving and departing from new stations, and neither the journey nor we ourselves are quite the same. Each age and station develops a different answer to the questions of who we are and where we are headed" …What Really matters (Kleinman)

As discussed and justified in the preceding chapters, in order to improve the software development situation in Ethiopia, better use of methodical approaches to software development needs to be explored. In particular, the selection, contextualization, continuous adaptation and use of appropriate (to the situation) methods and processes for software development are believed to be critical factors in the effort to bridge the oversize demand-supply gaps observed.

The approach reported in this chapter is one attempt in this direction. Drawing on the works of renowned methodologists, industry best practices, and own experience on the one hand, and the findings and requirements reported in the preceding chapters on the other, an attempt is made to develop a flexible and context sensitive approach. The approach developed, 'Reflective Steps", is for the most part an enhanced version of the STEPS approach originally developed by Floyd and her co-workers (Floyd et al., 1989). However, it also draws on: the Reflective System Development method proposed by Mathiassen (2000), Boehm's Spiral model as modified by Pomberger (2006), and the agile method developed by Cockburn (2006). Reflective Steps is an iterative, collaborative and learning-oriented software development and process improvement method, particularly suited to such immature software engineering environments (such as the one in Ethiopia). It is also a distilled version of years of experience – a pattern that the researcher has observed to work well over the years of professional practice and experimentation.

In the remainder of this chapter, first the premises on which the proposed approach is based are explained. This is followed by a general description of the Reflective Steps approach. Remainder sections of the chapter are devoted to providing detailed descriptions of the proposed approach.

5.1 The Premise

In this section, first the thesis for Reflective Steps is explained. The explanation also includes how the thesis is mapped to the proposal. Next theoretical perspectives and related works that served as additional sources of inspiration in the course of developing the Reflective Steps approach are briefly presented. The last part of the section presents the original STEPS model which served as a reference model for Reflective Steps.

5.1.1 The Thesis

The Reflective Steps approach is premised on the following thesis. For clarity, the thesis is paraphrased in a number of paragraphs.

A software development work is a highly collaborative activity carried out by groups of people in various roles (software engineers, application domain and process experts, user representative etc.). It involves interleaved processes (of project management, software production and embedment into an organization) that are carried out within given productivity and quality constraints. It operates in a continuously changing contextual situation[10] (people, technology, business processes, perceptions, etc. factors that influence the software development process are all subject to change during the projec).

Under such circumstances, predetermined methods and practices do not help in achieving the desired productivity and quality constraints in a specific development situation. Instead, the process (or the journey to the destination) needs to be carried out step by step

[10] This is in contrast to the traditional thinking among the software community where the understanding of change is dominated by 'change in requirement' which mostly referred to change in the business process. From the experiences reported we have seen how change in management, change in technology, change in the perception of actors, etc. affect software development.

incrementally, each step resulting in delivering a portion of the product as a proof of user requirement and design concept through iterative cycles.

In the move forward through such steps, as much as the prevailing context at the beginning has already determined the development strategy considered for the step taken, actions taken during the steps also affect the context – the context changes. The experiences resulting from the actions taken provide opportunities to better understand the application domain, the development process, capabilities and related tradeoffs. These in turn give more chances to revisit plans and development strategies for the next steps forward.

Accordingly, for software productivity and quality to be improved in subsequent steps, process and progress experiences as well as feedback on the product delivered at the end of each step should be used in refining and adjusting strategies, balancing and integrating contextual constraints as well as targets. To make sense of (learn from) such experiences and feedback, collaborators in the process need to collectively reflect before, during and after each step and at various levels (individual, group, project and organization levels).

As such, the software development situation calls for taking 'reflective steps' on a regular basis to incrementally develop and deliver the product required. In the process, specific processes and methods suitable and usable to the specific settings are discovered by continuous adaptation of the methods and process in use based on project experiences in earlier steps. As such, project groups and organizations can use iterative/regular 'reflective steps' as a learning, communication and negotiation mechanism to be used in software design, software project management, and software process improvement. Such 'reflective steps' can also be used for software competence building by using experiences strongly grounded in the actual software development project. Such is the essence of experience-based learning at each step to continuously improve the process of development to deliver the product required. With this approach, not only an aspect of a product is quickly and continuously delivered to users, feedback on the process and performance is quickly and continuously provided to developers to adjust strategies and improve performance.

Mapping the Thesis to the Proposal

Given a software project, the overall strategy adopted to increase the usability of methods and processes in the effort to develop usable software is one that involves:

- finding an appropriate approach for the context,
- devising a mechanism to continuously adjust the approach to fit the changing context, and
- building the knowledge of professionals on the approach, its adjustment and use.

According to the strategy adopted for this research, finding an appropriate approach for the context involves:

- understanding of the context,
- selecting a suitable approach from those publicly available and widely used approaches in the industry, and
- tailoring it to fit the context.

Based on the knowledge about the software development situation in the local setting as well as the potentials and limitations of existing approaches, and years of experience in using and experimenting with methods, as established in previous chapters, an attempt is made here to:

- introduce a suitable method for the local setting by adopting and tailoring one (or combination) popular method(s) in the industry;
- provide a mechanism through which the approach is continuously adjusted and improved by the practitioners themselves to serve their purposes in the course of developing a usable software. The involvement of practitioners in the process in turn provides an opportunity for them to build their knowledge by learning from experience; and
- introduce a learning mechanism by adopting and tailoring popular experience-based learning models.

To provide an opportunity where the experience gained in the project is used to better meet the quality and productivity requirements of the project, the software development is planned for execution at incremental steps. To benefit from such experience at each step, we must learn from the experience. For this purpose, reflection is proposed as a mechanism of learning from experience. This is the justification against which 'Reflective Steps', an approach that combines step by step project execution and a reflective learning mechanism to benefit from experience in the project to improve performance and quality is proposed.

To demonstrate the foregoing, an attempt is made to revise the STEPS process model as follows:

- to include and explicitly address important software development activities that strongly feature in a typical project but do not seem to be adequately addressed in existing process models, and
- to support a step by step approach for software development and software process improvement; where software engineers and users work together; where each step follows a design-critique-reflection (experience-based learning) iterative cycle:
    - to produce a software increment
    - to ensure usability of the increment
    - to adjust the project plan
    - to improve the software process

These and related aspects of the proposed approach are further discussed in sections beyond providing a brief background on theoretical perspectives and the original STEPS model.

5.1.2 Theoretical Perspectives

What follows will provide further background and justification to support the proposed thesis and approach in terms of briefly reviewing theoretical considerations and philosophical underpinnings published in related literature. To render meaningful discussion within the space limits of this chapter, only aspects related to reflective practice and participatory design in the context of software development are briefly reviewed. Additional review on reflective learning is provided in Chapter Six.

(i) Software Development as a Reflective Professional Practice

Software development practice is a particular instance of reflection-in-action.

Review of related literature (Schön, 1983; Mathiassen 2002) reveals the possibility of applying two different perspectives to professional practice in general and software development in particular: technical rationality and reflection-in-action.

From the technical rationality point of view, professional practice is seen as instrumental problem solving. In this context, the practitioner starts with given objectives and chooses optimal means to realize them. In doing so, the professional practitioner uses scientific knowledge to perform specific tasks and to select methods and techniques that apply to different types of situations within his/her practice. According to this view, professional practice situations can be categorized scientifically, the knowledge applied is a result of scientific work, and professional practice is seen as different from related scientific works. That is, professional practice applies scientifically-based theories and techniques whereas scientific work develops these theories and techniques.

In contrast, the reflection-in-action perspective assumes that the different situations of professional practice are unique, complex, uncertain and even discordant. Here the practitioner must be aware of the uniqueness of the situation and behave accordingly. In addition, it is often only possible to see or comprehend small fragments of the situation because situations are typically dynamic, consisting of complex networks of problems and conflicts. According to this view, knowledge and action are intrinsically related. The practitioner's knowledge is in his/her actions and cannot be fully described. Professional practitioners do research in the situations they find themselves in, reflecting while they act, and developing new insight as part of their daily practice.

Most of the conventional literature supports that technical rationalism (and the methods on which they are based) is a more realistic or useful view for software development. What is needed from the technical rationality point of view, is a suitable set of general methods and techniques for the systems, combined with methods for the analysis of situations and selecting of appropriate methods and techniques (Davis, 1982). These methods or the processes thereof do not provide concrete guidance and actionable items where problems are encountered following existing (defined or improved) processes during a development project. For instance, when a project manager identifies a schedule overrun that threatens timely project delivery, or a certain artifact design fails to work, methods hardly give any specific recommendations or guidance to get the project back into schedule.

There are also strong counter-arguments that the use of methods and formalization alone cannot always help in reality (Mathiassen, 2002; Bjerknes et al., 1990). These workers suggest the importance of experiences of the practitioners and their reflection on the situation at hand as well as the adaptation of a combination of formal and informal approaches. From the reflection-in-action point of view technical rationality alone (or methods thereof) is far from sufficient.

> In general, the ways methods are used in practice differ from the ways methodologist describe them for use. Methods may be understood as guidelines for practice and expressions of espoused theories on systems development; methods-in-use may be understood as those parts of system development practice in which practitioners apply methods to practical problems; methods may be understood as non-canonical practices which are not (or cannot be) explicated or codified as the others (Mathiassen, 2002; Bjerknes et al., 1990).

Accordingly, system developers must know more than mastering a repertoire of general methods and tools. They must know how to cope with the specific environment in which they work. They must open their minds and engage in reflections and dialogues, with themselves and the environment, to generate the necessary insights into the situation at hand. In this sense, software development practice can be seen as a particular instance of reflection-in-action.

Creativity and Social Discourse are requisites for software design

Design (the process) is essentially a social practice, and design (the product) is a social construction often carried out in a collaborative work environment (Mamykina 2002, Floyd 1987). Because of the abstract nature of software, we can learn and communicate about it in personal discourse with experts (program designers, users), by defining documents and by trial and use. Social discourse among stakeholders is crucial in software design. In the collaborative software design settings, where pluralistic and meaningful social discourse among stakeholders is supported, the development of mutual understanding, common language and an ability to communicate and exchange design ideas are essential in the design process.

Individual creativity is essential for software development and as such needs to be nurtured and encouraged. Such individual efforts ought to be enriched and augmented through interpersonal interaction, debating best solutions, and reflection on both the content and process of the design to arrive at an agreed solution for general adoption. As such, design involves co-construction of knowledge, commonly upheld by collaborative sessions. A typical collaborative session may involve presentation of a discussion object[11] followed by a critique and reflection session (design-critique-reflection). The presentation sessions provide an opportunity to share and clarify ideas, understanding and design artifacts. While one group presents its work, others question, constructively criticize the work presented with the view to test and enrich it. In the process, convergence and mutual intelligibility of perspectives can be attained through ongoing discussions, reflections and negotiations among stakeholders. From this perspective, design could be considered as inherently a collaborative reflective activity/process.

Under the existing circumstances, software design projects may fairly be categorized as ill-structured and wicked projects in the sense of Rittel and Webber (1984) and Budgen

[11] In the area of software development, the objects of discussion could be: service contracts, project plans, design artifacts, process and methods, feedback on software use, etc.

(1999), as they have become projects that deal with problems that can't be resolved with traditional analytical approaches. As Budgen wrote, in a wicked problem, a solution’s different aspects are so extensively interconnected that in adopting a particular solution to any one part of a problem, the resulting interactions with the problem itself might make the task of solving it even more intractable. Although this original concept arose in the context of social planning, many characteristics of a wicked problem (such as the lack of a stopping rule and the absence of true or false solutions) are readily recognizable facets of software development. On this basis, Budgen asked, how do we stop pretending that designing software is largely a matter of following a set of well-defined activities, and recognizing it as a creative process that requires us to find ways to develop the design skills needed to build the software systems of the future?

The consensus, nowadays, is that software professionals need to be able to learn from reflection as the problems they face are often unstructured and novel, with multiple possible solutions.

In this connection Schön’s theory of design is worth noting Schön (1983). According to him, designers work in alternating cycles of action and reflection. The designer *acts* to shape the design situation by creating or modifying design representations (such as papers, mockups, or computational artifacts), and the situation ‘‘talks back’’ to the designer, revealing unanticipated consequences of the design actions. In order to understand the situation’s back-talk, the designer *reflects* on the actions and consequences, and plans the next course of action. Such action results in modification of the design representation, and the action-reflection cycle continues. This action-reflection model illustrates the design activity of a single designer, and the reflection takes place in the head of the designer.

Reflective thinking is requisite for professional competence building

At another level, in a professional work environment such as software development that often results in introducing change in the organization, the challenge of making sense of daily work procedures, interpersonal interactions, and issues of power can be assisted by attention to reflective processes specifically designed for these purposes. In a related professional practice (the nursing profession), for instance, Taylor (2006) emphasized the usefulness of reflective practices as a means of facing and unraveling the conundrums of practice and learning from the insights gained. Taylor suggested three processes of reflection to assist nurses in building competence in their daily work.

- Technical reflection: a reflective practice associated with task-related competence building that relates to testing the validity of existing work procedures, techniques and practices with the view to replace them with better ones. This relates to reviewing the situation (including analysis of issues and assumptions) based on information obtained from the process.
- Practical reflection: concerned more with human interaction or communicative patterns that are set up by nurses and the people with whom they interact. Mediated through language, and based on incidents at work, reflection involves experiencing (retelling a practice story so that you experience it again in as much detail as possible), interpreting (clarifying and explaining the meaning of a communicative action situation), learning (creating new insights and integrating them into your existing awareness and knowledge).
- Emancipatory reflection: involves interpretation of roles and social obligations in the context of politics and seeks to provide emancipation from oppressive forces that limit people's rational control of their lives and practices.

According to Taylor (2006),

> "humans are the only life form who can reflect on their experience. …Whatever has been known or can be known is a source of reflection. A view of reflection at these proportions makes it too large and complex to be almost unimaginable and unmanageable; thus effective reflection requires focus with practical and systematic processes."

Taken together, reflection is considered to be an important part of the learning process that targets professional competence building and there are many theories about what reflection is and why it is so important especially for learning from experience, developing the skills of professional practice and for the development of skills which are said to enhance learning. To this end, in addition to the foregoing account, more theoretical aspects about reflection, particularly about its application and use in experience or work based learning are presented in Chapter Six as part of the learning model incorporated in Reflective Steps.

(ii) Participatory Design

Joint Application Development (JAD), originally invented by IBM around 1977, is a group-based design process. Its general purpose is to bring together developers and users to jointly define an application. In practice, the JAD is a structured and facilitated meeting used for group input, discussion, and reaching consensus around requirements, plans and decisions. In a JAD sessions, a group of stakeholders is authorized to make plans and decisions, guided by a facilitator, and supported by people in specific relevant roles.

JAD is probably the best known class of application software design workshops in widespread use. But it is not suitable for all development situations. In this connection, Jones (1997: 24) wrote: "JAD is probably least effective when major organizational or team-building work is required of the project/stakeholder team. Merely gathering a group of loosely related systems consultants or developers with the target users does not guarantee a productive session. Trained and experienced facilitators are essential to producing worthwhile results, and a formal JAD should not be attempted if facilitation is not available. Also, if a creative process or a breakthrough product is required of the team, a JAD approach is typically too highly structured to lead to innovation of new designs. Finally, although JAD is consensus oriented, it is not designed to be democratic. JAD goals and scope are typically predefined in advance of sessions, and alternative methods are not readily available to practitioners."

Participatory Design (PD) is a design discipline that has its origins in Scandinavia in the 1960s. It is based on the principle of including users fully in the design process. Contrary to JAD, here users fully involve in all development activities (beyond simple attendance design workshops) including the sharing of responsibility with the developers for the quality and performance of the system developed. The type of participation is more of transformational in the sense of Dahms & Faust-Ramos (2002). One of the tenets of PD is that system workers should be given better tools instead of having their work mechanized. In this approach, users' perception about the technology in their work is as significant as the technical requirements for the technology.

After comparing JAD and PD, Carmel (1993) reported on the provision of mutual reciprocal learning in PD projects, wherein the designer team and users learn from each other about system design and work practices. In true participation, users and system experts share the responsibility for the quality of the design proposal and the implemented system. Thus both system experts and users get new roles in the system development process. The system experts cannot make final design decisions on their own (Bjerkness, 1993). PD also provides opportunity for design by doing, wherein users have direct hands-on experience in the design works. The use of low-tech design objects that are familiar to all will help in bringing the users directly into the design process. When the system is implemented in an incremental way, users can see progress that encourages them to increase their participation. After all, the most important part of the project is when a system is made usable, and that is the time the users do most of the work.

Another important software development process derived from PD is the Contextual Design (CD) developed by Holtzblatt and Beyer (1993). This is a design methodology for engaging users' participation in the design of their systems and tasks by fully involving their work experience. By interviewing users in the context of their work environment, and treating them as the experts in the work processes to be designed, a contextual view of the system is developed in CD. In this approach, as compared to other traditional requirements gathering approaches, much of the control for information collection is placed in the hands of user (Jones, 1997). Contextual inquiry is typically conducted as a set of processes adapted to the environment. As such the user context is the basic model for contextual design in the way that JAD is a model for the business context.

The Reflective Steps approach proposed in this research draws on these approaches, but with more emphasis on transformational type of participation as suggested by Dahms & Faust-Ramos (2002). In particular, the type of participation proposed is collaborative, instead of passive participation) which is rather similar to that of the Team Design Approach of Jones (1997). It is based on the process of mutual learning experience between users and software engineers as reported in Section 4.4 of this report. The user is put on equal plane with the system designers; responsibility for the product is shared; and this is made clear from the outset. In this role, in addition to providing information and feedback, users agree to provide access to their work, access to their insights and rich operational experience (as in the Operation Group). The software engineers bring their technical design expertise to the project. As Bjerkness (1993: 39) wrote, "the users must learn about technology from the system experts in order to understand what computer technology can do for them, and the system experts have to learn about the application domain from the users in order to build a flexible and efficient system that fits the users' needs." According to Jones (1997), such an approach is useful for any major organizational change where systems and processes are redesigned, and the users' jobs or work processes will be affected. As much as users are given the responsibility to redesign their work processes, attempts are made to introduce best practices through the envisioning activities supported by software specialists and external experts (as in the Critique Panel).

5.1.3 The Base Model – STEPS

The main approach selected for adaptation in this research is the STPES approach to software development with users, proposed by Floyd and co-workers in 1989 (Floyd et al., 1989; Floyd, 1993). STEPS is a methodological framework that promotes evolutionary software development methods with emphasis on participative communication & learning process, usage-orientation, organizational embedding and versioning. Because of its emphasis on framework, it does not directly provide specific techniques and tools for its activities. Instead, the use of applicable techniques from other methods is recommended. As of recent, however, methods both at social and technical levels are being developed based on the concepts and principles of the STEPS approach. One such attempt by Wetzel (2001), for instance, relates to the development of Anchor (Anticipation of Change in Organizations) which advocates the use of anticipation of change as a design rational. Anchor provides user-oriented representation techniques, together with the definition of application kernels & system stages. It also allows switching between the workplace or workgroup, interdepartmental and business perspectives in the course of the system design work. Extending further the STEPS concepts and principles of organizational embedding, evolutionary software development and use context, the Tools & Material (T&M) approach (Züllighoven, 2003), attempts to incorporate/consider technologies and tools used in the real world work environment into object-oriented approaches to application software development. In the T&M approach, application-orientation and usage-orientation aspects are more emphasised in operational terms.

Figure 3 below depicts the structure of the original STEPS model.

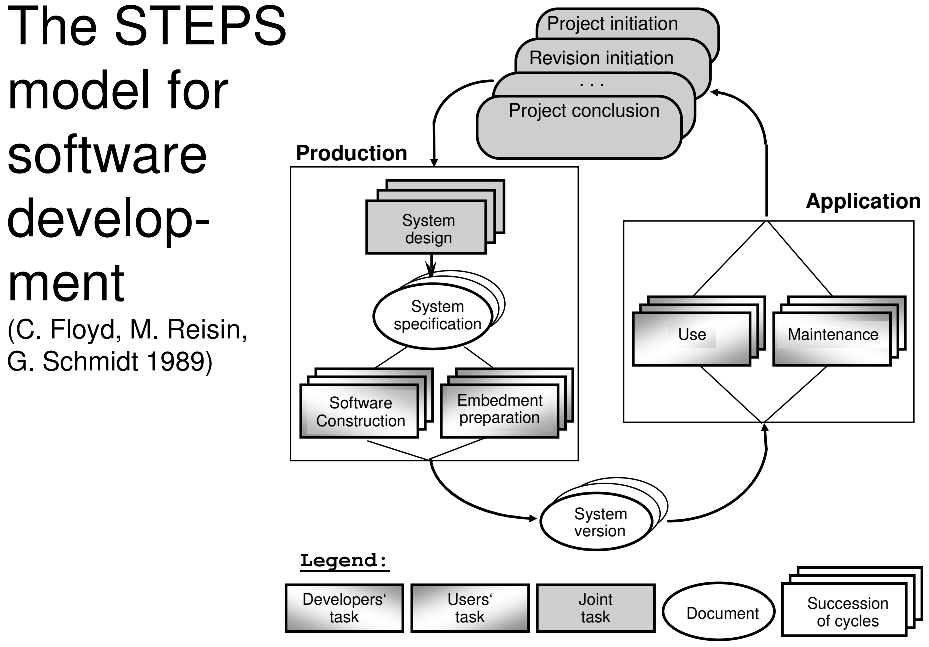


Figure 3: The STEPS model for software development

According to Wetzel (2001), the emphasis and aims of the STEPS model and its extensions lie:

- in evolutionary software development based on a cyclical process model;
- in support of participative communication and learning process for developers and users alike;
- in the emphasis on the use context, which results in an interlacing of system design and organizational development;
- in a task oriented requirements analysis, oriented to the tasks of organizations instead of system functions; and
- in the support perspectives, which is expressed in the leitmotif of software workplaces for qualified human activity and the user as the expert.

STEPS advocates collaboration in design as a means to improve quality. Quality in the STEPS philosophy is often associated with processes of using the software developed. In this connection, Floyd (1987) wrote:

> "to determine the quality of computer use in organizations, we need to consider issues concerned with qualification, motivation, training, work organization and traditions of use, together with the features of the product [software] itself".

STEPS assumes that processes of computer use are highly creative to the extent of developing work practices that enabled users to work around known software shortcomings.

> "To bridge the gap between the functions of the data processing systems and their actual work-tasks, user communities developed practices that can be described as 'fitting' system functions to actual needs, 'enhancing' system functions by mutual practices by manual practices, 'working around' inadequate system functions in order to achieve human work-goals"(Floyd 1987: 249).

Floyd (1993) further explained that,

> "Crucial for judging the quality of computer-based systems from the users' point of view, is an evaluation of the quality of their work when using the software. This gives rise to our view of software development, which is not restricted to the product software, but also takes into account the social processes and relations in the context of which it is produced and used"

Among the quality assurance techniques suggested for consideration to cope with the quality gaps experienced by users and designers, were: reviewing design in teams, establishing roles in development teams, prototyping, and role-sharing between users and developers. According to Floyd (1987), "in order to make these techniques fruitful, designers must ensure that software systems and underlying design decisions can be modified as user needs become better understood. They are then able to enter into processes of communication with users, which step by step, help to tune data processing systems to actual user needs". Here seems to lie, according to the author, the origins of the 'people-orientation' movement, the likes of those that are widely promoted and advocated by such newly emerging software development methods as Agile. Arguably, the 'people' in the Agile methods refers to the technical people involved in the development. Under such reference, the author is hesitant to acknowledge whether the issues raised in the 'people' dimension in the Agile process fully accommodate user collaborators in the real sense. The contention here is it should! User representatives in Agile methods, however, are still perceived as task specialists or support to the main development team (for more on this see also Section 5.3.4 of the report).

In addition to the forgoing, perhaps most of the reasons behind choosing STEPS as a base model for the current work are captured in the following concluding remark by Floyd (1987), [emphasis added]:

> "giving priority to the process-oriented view [which is the emphasis in the current research] would imply dealing with conflicts and contradictions which are abstracted from the product-oriented [which is the traditional technically-oriented mechanistic approach] view. It would require us to change our attitudes towards those who will use our products; to think of them as partners in spite of conflicting interests; to learn to give and take criticism in a supportive and constructive manner; to learn to work in technology keeping in mind human values and changing human needs. It would mean going beyond the mechanistic world-view embodied in the product-oriented perspective."

In fact, the title of the publication by Floyd (1993) says it all: "**STEPS to** software development **with users**" [emphasis added]. The following are additional quotations from this seminal publication presented as further support for the choice to use STEPS as a base model.

> "While software developers are directly concerned with programming, they contribute indirectly to profound changes in the work life of the users of their programs"

> “Users, on the other hand, are faced with a revision of their work processes. This involves re-organization of work, the acquisition of new skills in using computer programs, and far-reaching changes in competence”
>
> “In keeping with this, we consider software development a learning process for both developers and users”
>
> “On the one hand, requirements evolve because of general changes in society, the economy, technology, law, etc. On the other hand, changes in work organization, users qualifications etc. which are not least an effect of the software system itself, give rise to new and changing requirements”
>
> “[STEPS] refers to a class of possible development strategies allowing the choice of a situation-specific strategy as needed in the project at hand rather than depicting one ideal development strategy to be copied as closely as possible in all projects”

As can be seen in the descriptions and explanations provided in subsequent sections and chapters, most of the concepts and principles that underlie these statements are the keys in addressing most of the problems being experienced in the existing situation. These issues are also considered critical in augmenting the application and performance of existing methods and processes to address the problems identified. Taken together, the STEPS insight: its cyclic, evolutionary and small implementation steps concepts, its emphasis on communication and learning-based design process, its recognition of the importance of the usage and organizational embedding process, are all in line with the essence of the Reflective Steps thesis described above.

5.2 Reflective Steps: Overview

As briefly explained in the preceding sections, the original STEPS model proposes development cycles consisting of (i) project or revision initiation, (ii) production, (iii) releasing a system version and (iv) application of the version. To address the contextual issues and concerns discussed in the previous chapters, the original STEPS model is revised by,

- introducing additional enhancements to accommodate project design related front-end activities to the software production process;
- explicitly addressing software development and delivery in increments;
- introducing reflective learning for continuous product and process improvements;
- introducing a form of collaborative development team building and work practice.

In this effort, the original model is recast into an integrated model depicted by the context diagram shown in Figure 4. This is further exploded into the process models depicted by the diagrams shown in Figure 5, Figure 7 and Figure 8. The corresponding Learning Models proposed are presented in Chapter Six.

This section explains the context diagram to provide an overview of the approach proposed. Subsequent sections address further details of the proposed approach by providing similar explanations for the process models generated from the overall model.

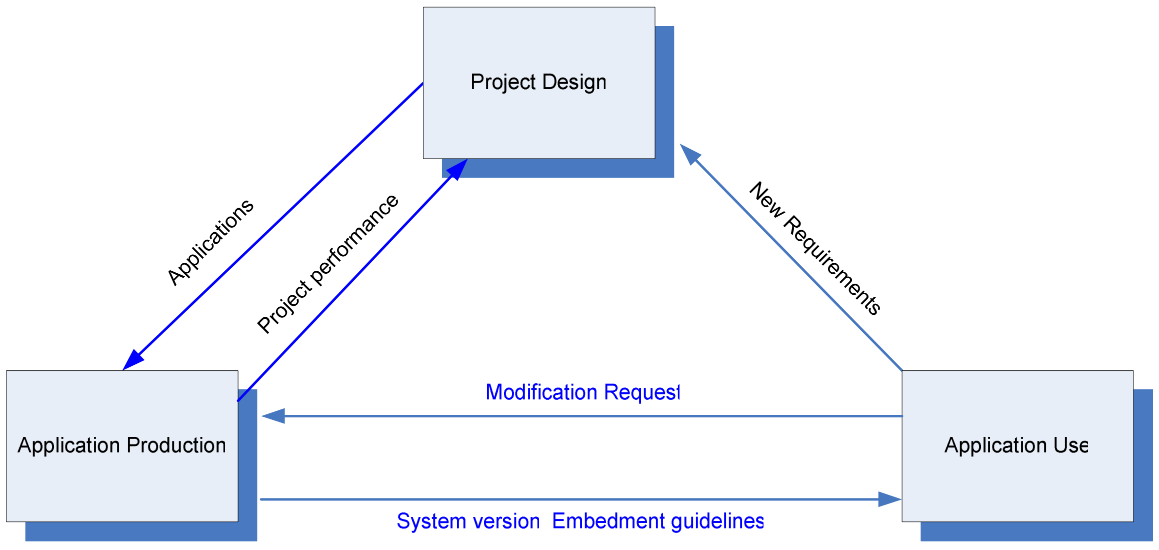


Figure 4: Reflective Steps - Context Diagram

The model shown in Figure 4 seeks to explicitly integrate the main aspects of any large project that specifically aims at custom building a software system for a given organization. As can be seen, it identifies three major sub-processes of the integrated software development process. Namely: Project Design process, Application Production process and Application Use process. In this connection, it is relevant to note the following. In Reflective Steps, each project is assumed to involve the development of more than one business application software (this is how the existing situation is characterized in Chapter Four of this report). In other words, a project with more than one business application is delivered incrementally application by application. The integration of the applications is maintained through the design of appropriate interfaces and architectures. As such a small project with one application is treated as a special case of this understanding.

Project Design defines the project, continuously tracks its progress and makes necessary adjustments in the project environment and constraints. In the process, it identifies and recommends business applications for development by the Application Production, receives performance related feedback from the Application Production process to reprioritize and revise the scope of outstanding applications. It receives new requirements for inclusion in the project from the Application Use process. The main actors in the Project Design process are management and planning experts.

Application Production focuses on the design and construction of the required application software (by working jointly with users) and within constraints defined by Project Design. It provides usable software to the Application Use process incrementally. It provides feedback to Project Design based on postmortems at the end of each application implementation. The main actors in the Application Production process design are software engineers, domain experts and key user representatives.

Application Use ensures that software versions delivered by Application Production are smoothly deployed and put to use in a sustainable manner. While it provides feedback in the form of modification requirements on the software versions delivered to Application Production, it identifies new requirements for consideration by Project Design. The main actors in use design are users, change agents and software engineers.

Although we have artificially reinforced their separateness in order to render meaningful discussion, these components are cyclical and interwoven processes involving at times overlapping activities. These issues are explained in the detailed descriptions provided in the following sections.

5.3 Project Design

As the Project Design process is one of the major enhancements to the original STEPS, an attempt is made here to provide an elaborate account on this process. The detailed activities involved in this process are depicted in the diagram shown in Figure 5 below. As can be seen, a spiral diagram is used to depict the activities. This is different from the STEPS diagramming convention adopted for Reflective Steps as can be seen subsequent sections of the remainder of this chapter. The spiral diagram is preferred for this component to accommodate as many of the activities as possible within one diagram and more importantly to clearly show the iterative reviews and reflections involved in the process. A version of this figure with the STEPS diagramming convention is found in Section 5.4. The main activities of this process are explained in the paragraphs that follow.

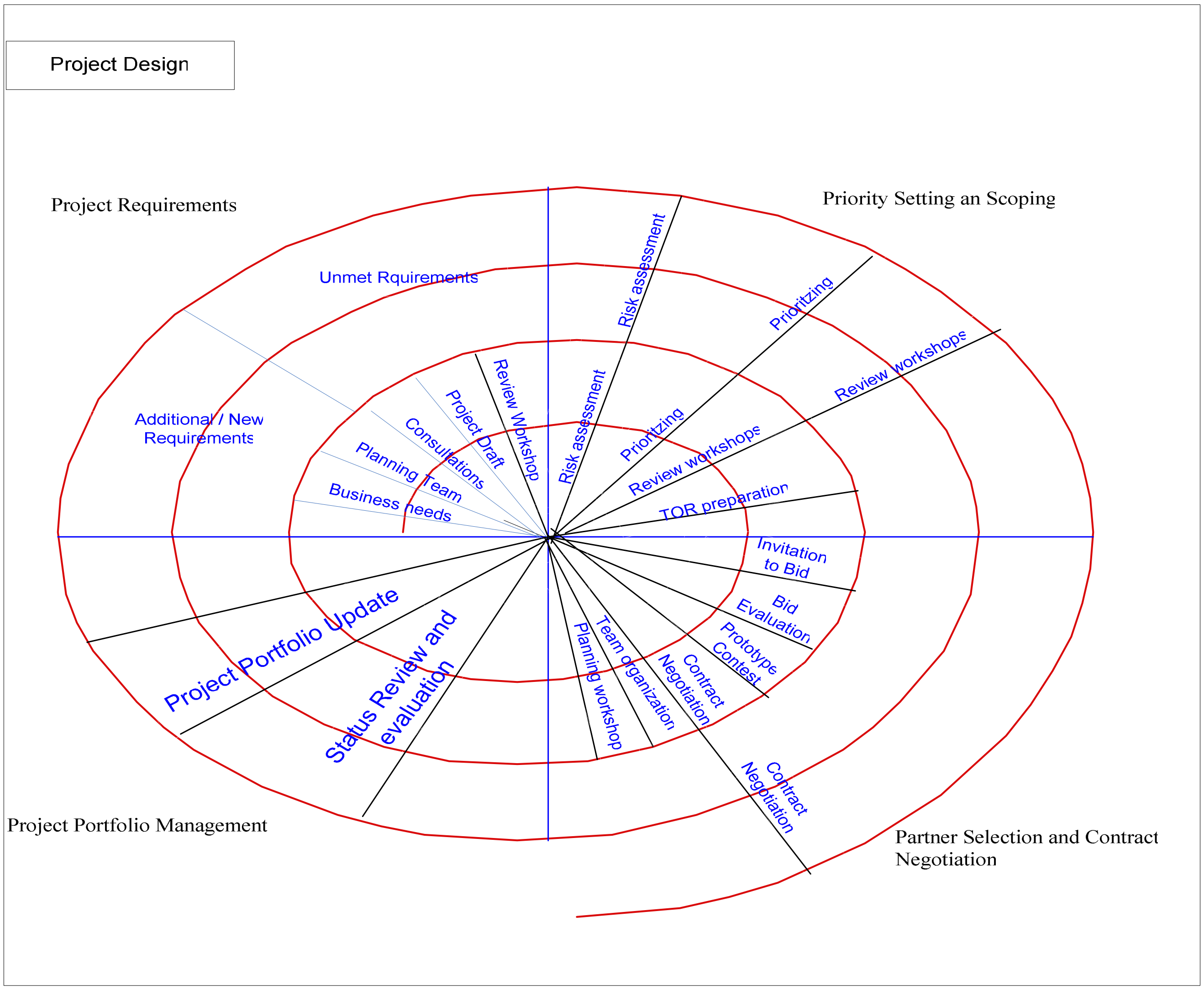


Figure 5: Reflective Steps - Project Design

As shown, among the main activities in the Project Design process are: project requirements setting, priority setting and scoping, partner selection and contract negotiation, and project portfolio management. Aspects of these activities are discussed below with more emphasis to those activities which are critical to the local situation but not explicitly and adequately treated in other process models.

5.3.1 Setting Project Requirments

The starting point of software development projects is often a request from the various business units. Such requests, after going through some sort of management approval process, are usually articulated in an assignment brief from management to the planning or IT support units. The usual practice in the local context is then to prepare terms of reference (TOR) and engage software houses for the job. The consequences of such practice have been discussed and documented in Chapter Two and Chapter Three.

> As established in the review of local cases, the assignment brief from the management is a necessary but by no means a sufficient condition to guarantee successful initiation and then deployment of the software projects in organizations. Particularly, in the local context, such authorization would only indicate the interest and intentions of the sponsor and lasts as long as the manager or the sponsor is in that position. Where managers or sponsors leave the organization or lose authority (which is often the case in today's Ethiopian reality), often the projects are discontinued. There seems to be a tendency to associate the project with the initiators. The design of the software project benefits more if such perceptions are brought to surface and explicitly addressed early enough in the project design. In practice, where this was done, better ownerships of the projects as well as partnership were observed – it was possible to create buy-in.

In short, an assignment brief essentially could be considered as the management's viewpoint or intervention into an organization. In every large project initiation, as evidenced in previous chapters, it is likely that participants and users have different viewpoints regarding the purpose or mission of the project. And usually this is the source of most of the misunderstandings and complaints that gradually grow into serious resistance and conflict at a later stage. As such, every large scale software project with the potential to bring about change in the user organization would benefit if it starts with an exercise that aims at some form of organizational understanding and learning. As part of this process, the roles, norms and values inherent in the organization need to be examined, and the disposition of power explored. An overall sense of the forces at work around the project need to be created. To carry out this activity, a small planning team may be constituted from experts in the planning unit, IT unit and concerned business units.

Such an exercise should be performed through conducting interviews and planned discussions with selected elements of the stakeholder community. Informal surveys of opinions and consultations at the grass roots level can also help. In the process, special attention needs to be paid to involve the middle managers who are going to be in direct contact with project activities. These groups often play the role of a gate-keeper to the flow of information up and down within the organization. Their involvement and buying into the project objectives and activities would facilitate smooth project operation.

In a similar undertaking with the business process redesign experiment conducted in Organization C, as reported in Chapter Four, we created forums where we elicited stories and critiques from users helped a lot. The stories, mostly expressed in the form of complaints, history and fate of similar past experiences, what worked and did not work in the past, etc. served as good starting points. For better and formal methods, see Boehm (1994, 1998).

The results from such an exercise are to be articulated and presented back to management and sponsors for more insight, discussion, feedback, reflection and debate, by hosting workshops. As required such discussions and reflections supported by further inquiries may be extended for several days.

Such analysis done properly usually result in a thorough and insightful evaluation of the organization's readiness for the planned project. It helps to sense early in the project the various viewpoints as well as potential barriers that are likely to block the successful initiation. It will also help to get project ideas gradually diffuse within the organization. The findings from such exercise have to be used in the project design phase, particularly to incorporate strategies that would help address the potential problems sensed, to mobilize support for the project, etc. The overall purpose is to get the various perceptions around the project shared, discussed and negotiated to lead to an agreed upon (consensually decided) reconstruction of the project perception and assignment brief.

As part of the reconstruction of the project brief, a shared vision and basic scope of the project, potential applications to be considered in the project, project constraints, risks and priorities may be identified for review and enrichment in a series of reflection workshops. The project ideas so discussed are developed into the definition of the project and development of a term of reference for its execution. Assignment of a project management body - composed of management, user representatives, internal IT experts, external advisor (consultant), to formally establish the project based on the deliberations so far - may be required. Such a body may also serve as a steering committee, once the project development team is established to take care of the day to day project execution and management.

5.3.2 Setting Priorities and Portfolio Management

A software development project may involve the development of one or more application software. Where this is the case, the business applications may be prioritized based on contextual constraints (organizational, financial, technical, etc.). This process may involve serious negotiation among stakeholders. To support this process, there is a need to develop and agree on a set of criteria for valuing application projects in order to prioritize them. Decisions based on these criteria are likely to be more acceptable to most elements of the organization if the criteria are developed with the input or review of as many stakeholders as possible from within the various sub-units of the organization. So typically, broad and organization-wide discussions of the criteria are to be held before they are finalized. While valuable techniques that help in this connection may be found in general project planning and management methods, one such method specifically tailored to software projects and worth exploring is the Theory W proposed by Boehm (1989).

> With the introduction of the WinWin Spiral Model (Boehm, 1998), which extends the original model by adding activities from Theory W to the front of each cycle, the model provides risk-driven and negotiation-driven approaches to the management of software development projects with emphasis to activities in the project design sub-process. Stakeholders also come together at the end of each application cycle, to review performance and plan the next cycle. Based on the

WinWin Spiral Model, further attempts are also being made to extend this model to address the critical needs of all stakeholders including those in the software use category through the development of Next Generation Process Model (NGPM) and support system (Boehm, 1994). See also Section 5.3.3 for further extension of Boehm's original spiral model adjusted to accommodate make-or-buy decisions.

To facilitate iterative reflection, it is suggested that a project portfolio be maintained to keep track of the negotiation details (the options explored, the factors considered in deciding the scope and priorities, etc.), the progress and performance of each application project. This requires, among others, to define each project in terms of such details as project name, objective and scope, start date, estimated duration, estimated cost, strategic value and priority, and so on for entry into the portfolio[12]. As shown in the context overview diagram (Figure 4), the portfolio is updated based on continuous feedback and post-application evaluation from the production cycle (see below). To maintain such information, an additional entry for 'performance' should be included.

According to Reflective Steps, the project portfolio has to be reevaluated by the management team on a regular basis through collective reflection sessions to determine which projects are meeting their goals, which may need more resources and support, or which may need redefinition of scope and priorities, which ones should be cancelled, what sort of contract revision is required, and related analysis and decision. Since the circumstances of each project and the business environment can change rapidly, a quarterly review is suggested at this level.

In this connection, it is important to emphasize the need to avoid information overload at management level (for that matter at every level). In particular, the details of each project should be kept at the project team level, administered by the individual project managers (for this purpose, the maintenance of a project journal is suggested as detailed in sections beyond). Only key information should be rolled up and presented at each level within the organization as appropriate. At the top level, what is usually required is to provide a summary of performance,

[12] A simple database may be designed to maintain the portfolio electronically.

progress report, a measurement of estimates against actuals and costs. In addition, as required, the steering committee may initiate a special project evaluation process at any time to learn and know more about the status of projects.

5.3.3 Partner Selection & Contract Negotiaiton

Activities discussed under this category include: the make-or-buy decision and related processes, contract details and planning.

(i) Make-or-Buy

An organization may acquire a software product in any one or combinations of the following: in-house development, outsourced custom-development, or customization of commercially available software. Based on the analysis in the previous chapters, the basic assumption here is that most software development projects are outsourced. To come to such a decision, however, usually a buy or make analysis is performed (more specifically, outsourced or in-house development, to emphasize that what we buy is more of a service even in the case of customizing a commercially available software). According to Pomberger (2006), based on years of practical (real-life) project experience, even where there is in-house capacity for the development, better still is to let the in-house IT/Software unit compete on par with external software houses and if selected work on business terms as far as the project goes.

Pomberger (2006) revised the Spiral Model to address weaknesses and omissions in the original model in connection with some of the issues identified above. In particular, enhancements were made on the model to recognize and provide mechanisms of addressing the make-or-buy decision (extending the use of prototyping as a methodological instrument in the make-or-buy decision) and accommodating technological developments since the development of the original model. The revised model is depicted in Figure 6 below.

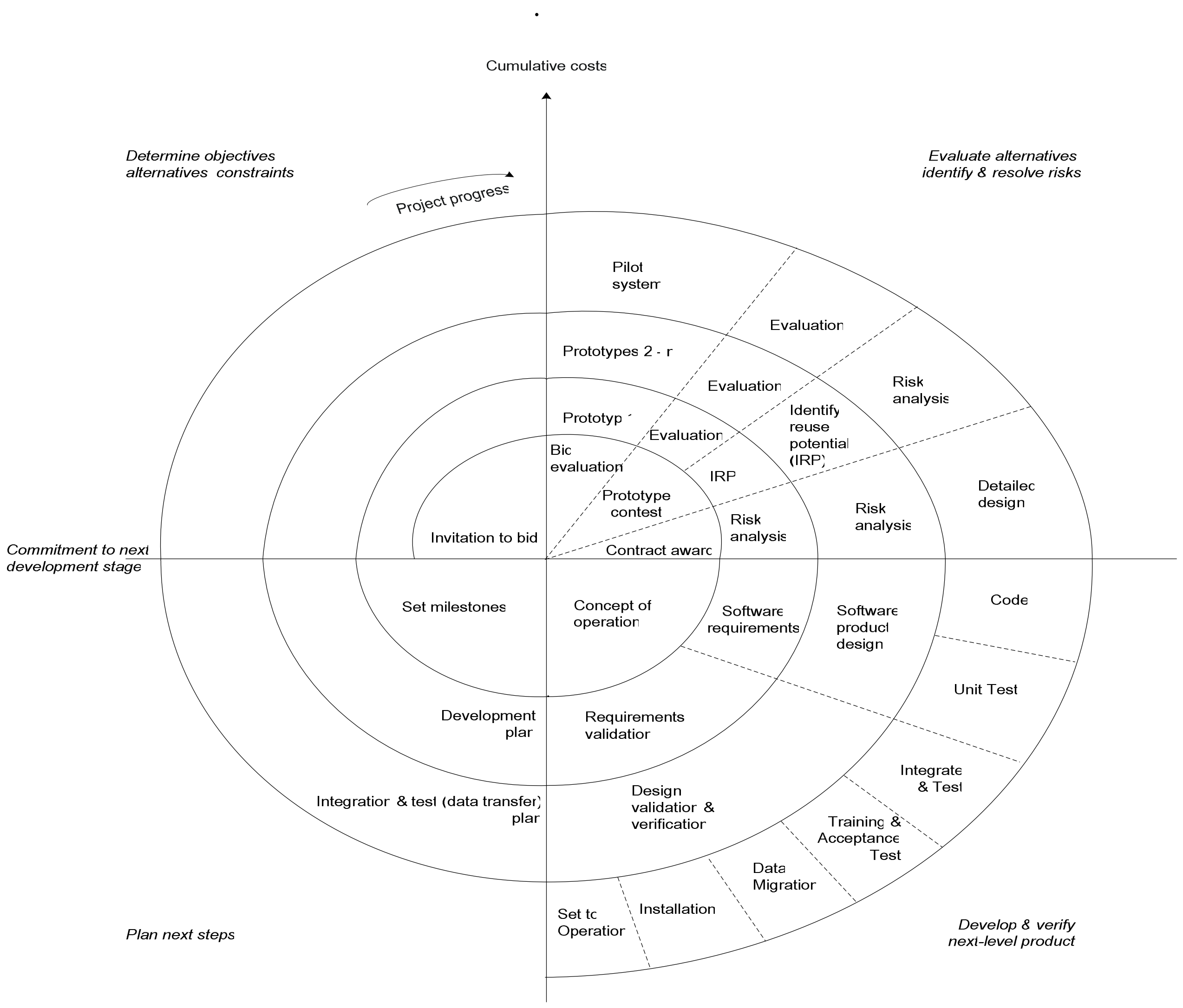


Figure 6: A Spiral Model of the software acquisition process (As modified by Pomberger, 2006)

In connection with software acquisition options, the revision proposed the following. Every software product acquisition should be made through competitive bidding (this also equally applies for in-house development). Two or three finalists (bidders) that have come out successfully from the evaluation process should be invited to a prototyping contest – to develop a prototype for one of the core processes that the new software product will support within constrained environment. The fees for the prototype contest are to be covered by the client. According to the revised model, experience from the prototyping will also be used for better project definition during the contract negotiations.

While fully concurring with Pomberger (2006) on the enhancements suggested for software acquisition, the author intends to propose the following further adjustments in order to increase the chance of successful implementation of the revisions suggested.

- Before the official release of the invitation to bid, having a consultative workshop with potential suppliers on the planned project may not only help to create a shared understanding around the plan, but also help to benefit from vendor inputs on related experiences and technological advances. The outcomes of such workshop may contribute to the viability of the bid document.
- To smoothly handle the situation in case the in-house group wins the tender on competitive terms, further details may need to be worked out in the areas of contract administration and enforcement, handling conflicts that may arise between employment and contractual terms and conditions of engagement, etc. Unless the group belongs to some sort of subsidiary company with properly defined terms of engagement, such issues and related roles and responsibilities need to be explicitly worked out as part of the contract negotiation process.
- With regard to documentation, the proposal in the revised model is to 'largely avoiding written requirements definitions' and using prototyping instead. In due consideration of the experience reported in previous chapters, a more realistic and appropriate option (at least in the local setting) would be to use prototyping as a major vehicle but support same by prototype-driven written documentation (instead of totally avoiding written documents).
- The author needs to emphasize the importance of the proposal in the revised model to engage an external consultant to serve as 'a project coordinator, arbitrator and conflict manager, with unrestricted decision-making powers'. However, there is also the need for organizational mechanisms to realize the authority and institutionalize the decisions of the external expert. For instance, this may be implemented in the form of membership of the external consultant in a steering committee, heading a formally established project office or project

management body, etc. Such an arrangement provides an opportunity to promote more stakeholder consultation and to benefit from collective wisdom to arrive at decisions. It also minimizes unnecessary accountability-related problems on the part of the external expert particularly in cases where the project mobilizes lots of resources and there are many actors that have a stake in the project or that may be affected by the decisions taken. It will also lessen over-dependence on or indispensability of the external expert. Definitely such support structure and arrangements are essentially necessary in the local setting drawing on the experiences reported in previous chapters. As Beck (2004: 76) put it, "the principle of alignment and authority and responsibility suggests that it is a bad idea to give one person the power to make decisions that others have to follow without having to personally live with the consequences".

(ii) The Contract

The make-or-buy activity involves among others the preparation of tender documents based on the details compiled that far on the project, including further elaboration of scope and requirements, followed by analysis of vendor offers to the tender, and then selection of a vendor or vendors.

Once selection of a vendor is made, the next activity is to negotiate the actual contract. Such negotiations are usually conducted based on the originally issued terms of reference (as published in the tender document), the vendor offers, the results of prototyping contest, recommendations of the tender evaluation group and new or changed requirements that resulted from developments in the business environment and organization since the publication of the original tender. This is required for various reasons. Often the time given to vendors to prepare the bids is short and inadequate. The requirements published in the bid documents and clarifications provided afterwards are generic by nature. These and related issues may have forced vendors to make assumptions about requirements that might have resulted in the under-estimation of costs and schedules. As reported in Chapter Three, vendors have also the tendency to offer cheaper rates for winning the bid, assuming that cost is the most important determining factor in the bid analysis. Taken together, such bid proposals are often driven towards winning rather than accurately portraying the size and complexity of the system and the effort required to build it. Accordingly, requirements, terms and conditions as well as proposals have to be readjusted towards reality during negotiation.

The actors involved in the negotiation process should include: user representatives, vendor representatives, project sponsor representatives, a consultant, etc.

As part of this negotiation process, it is also important to revisit the boundaries of the project scope based on the findings of the bidding exercise. Scoping is the process of defining boundaries for a project or system. It is an activity to be carried out jointly and collaboratively by users and the development team. As Jones (1997:180) stated, "boundaries are set by [users] to ensure that the work they support is not overextended or overbuilt, and boundaries are set by development management to ensure that a product is delivered within cost and schedule constraints". At the same time, it is also important to note that such boundary setting consists of many scope tradeoffs and negotiations (Boehm, 1994), and the boundaries set, redefined, and reset continually until a baseline scope is agreed upon.

Other issues that may be addressed during negotiation include: project organization and governance structure, schedule, deliverables, etc. Discussions on roles and responsibilities are particularly important in collaborative development environments. Some of these issues are discussed together with collaborative team development in the next subsection.

Although both users and vendors often believe that the negotiation and differences thereof are up to the signing of the contract agreement, in reality this is a continuous and iterative process. Changes are inevitable due to the unfolding learning process during the project lifetime – initially defined project priorities, scopes, contract terms and conditions, etc. evolve through iterations as problems are encountered and new possibilities are revealed. For this reason, it is not advisable to freeze contracts until the requirements and the design efforts are better understood. On the other hand, it is not affordable to leave them unnecessarily open indefinitely (as this has implications on resource management and utilization).

The outcome of successful negotiation exercise is a flexible contract and contract management process that provides for handling changes in a controlled manner, instead of the traditional and rigid contracts. Such flexibility of the contracts may also be realized through an iterative negotiation between the production team and the planning team or management, based on the feedback for project experience.

A flexible contract should, among others,

- provide for flexible planning;
- provide for a layered project governance structure with appropriate interfaces;
- create the necessary infrastructural support for effective communication, coordination and knowledge sharing; and
- provide the necessary environment to create collaborative development teams composed of users and vendors.

In addition to the rights and obligations of the participating parties, in a collaborative development environment, a contract should specify the amount of time the users can/shall spend on the project per week (Bjerknes, 1993). Otherwise, the users will give priority to their routine work tasks, making it difficult for them to do project tasks between meetings.

As part of the project governance structure, including a steering group (an upgraded form of the planning group discussed earlier) in which conflicts can be discussed is usually recommended. This is different from a project management group established at the technical level to take care of the day-to-day running of project activities. Sometimes the tension experienced by the different groups can lead to frustration and conflicts. In such cases, it is important to have a contract and a forum where the problems can be discussed. Having a steering committee for this and other purposes may contribute to make up for incompetence in project management (Bjerknes, 1993). Such arrangements give users and project sponsors the opportunity to seriously engage themselves in the provision of feedback, evaluation as well as steering of the project.

(iii) Overall Plan

From experiences so far, plans at every level are a must in the local reality. Without going into details to explain, taking note of the local realities reported in Chapter Two and Chapter Three, the author takes the position that: no plan means no project! As Boehm and Ross (1989) put it, we have to "Plan the flight and fly the plan; and, identify and manage risks".

Under the local circumstances, one can not underestimate the benefits that can be obtained from the planning process. Obviously plans are useful for such managerial activities as resource allocation, monitoring and control. However, more importantly in the context of software development, the process of planning (when it is done step by step, through active participation of all actors and supported by reflection) provides a unique opportunity to develop knowledge of developers in the application domain, the software process, and human interaction and communication. It is an important instrument to create a shared vision and understanding of project objectives, roles and responsibilities among the members of the technical team.

On the other hand, one should also take note of the fact that it is not possible to plan everything ahead, and that many activities are 'situated' with the actions that need to be taken contingent on factors which could not have been anticipated. According to Lucy Suchman's plans and situated actions, cited in Hall and Fernandez-Ramil (2007), every detail of work cannot be planned and made explicit, and in some circumstances at least people must draw upon their tacit knowledge to respond to situations as they arise.

As with methods in general, starter plans need to be developed and continuously adjusted throughout the project lifetime based on experience and learning resulting from its implementation. That is, one is in a better position to plan as one moves forward and gets feedback on the actual performance based on previous plans. Both the development and adjustment of plans should be done with the participation of stakeholders, and continuously throughout the project life time. Reflection becomes an important instrument to provide for such planning exercises.

As part of the plan, in addition to roles and responsibilities, reporting mechanisms between the various bodies of the project governance structure as well as the project stakeholders need to be formally established and practiced. Formal mechanisms for information exchange need to be clearly established, including the definition of forms and formats (report, template, etc.), frequency (how often, weekly, monthly, quarterly, etc.) and mode of exchange (hardcopy, softcopy, database, web, etc. via mail, meetings, etc.). Time and place for regular reflective workshops and meetings need to be established as well.

5.3.4 Collaborative Development Team

Another important aspect within the Process Design activity relates to the issue of institutionalizing collaborative approaches. As indicated in previous chapters, the most commonly observed mode of user participation in software development projects is ad hoc and assistant. In such arrangements, user representatives are called in to participate in the requirements definition and testing phases of the project. Often they join the project for a brief time and leave after providing the required information in respect of these activities. Even where seemingly permanent assignments are reported, the user representatives assigned are often seconded part-time to work on the project while still keeping their substantive posts with their employing organizations. In other words, their engagement and participation is not full-time. With such arrangements, motivation is low (Wetzel, 2001) and there is generally reluctance to participate actively and fully.

Emphasizing the need for collective participation in this connection, Wetzel (2001) wrote, “whoever is involved in IS development (in-house staff, consultants or vendors) has certainly to face and nurture collective participation and a great deal of negotiation within the whole process.” Particularly where there is a network of interdependence in the work area, “the anticipation (foresight) of organizational change caused by the realization of new design ideas and alternatives needs to be assessed, evaluated and improved during the design process by those whose work will be affected by the system's future use. It is crucial not to overlook interdependencies and the parties involved." As Erickson (1996) explained, design is not just about making things. Design is a social, collaborative activity. Gone are the days when one person conceived, developed, produced, and sold a designed object to others.

Software design is a complex phenomenon which involves an iterative process of requirements understanding, visualizing design to meet requirements, testing them, revising the design, testing, etc. An important part of the iterative process is collaborative work among designers and between designers and users. Designers can not simply study users' needs prior to the design phase and then forget users until it is time to test the system. Translating user needs during the process of design involves a great deal of interpretation, which can lead designers in directions which are not in the users' best interests. In other words, users need to be involved in the design process and design decisions for the design to move in the right direction.

For these reasons, in the process of system development for organizations, the support and engagement of users and each organizational unit are essential at every stage in the development and implementation. At the project design level, user involvement is a prerequisite for political decisions concerning the prioritizing of requirements, outsourcing decisions and the allocation of resources. At the production design level, user involvement is essential for decisions concerning design. At the use design level, user involvement is important to avoid resistance to change and for effective utilization of the system.

In view of the foregoing, the existing arrangement of development team composition, organization and work arrangement in software development projects needs to be revisited. The importance of collaborative development by fully and completely involving key user representatives throughout the project period were repeatedly emphasized by workers in the field (Floyd 1987; Mathiassen, 2002; Cockburn, 2006). However, actual strategies to realize same seem to be not fully developed yet. For instance, Floyd is among the earliest workers who strongly argued for such collaborative software design to the extent of demanding a paradigm shift in software engineering to accommodate this. Even though she raised important issues with strong supporting arguments, infrastructural support and mechanisms on how to operationalize the recommendations in practical terms were not explicitly explored. Mathiassen admittedly limited such collaborations to technical professionals in his reflective system development approach (while stating at the same time the need for extending same to include users). Even in the Agile processes (Cockburn, 2006), where the issue of collaboration seems to be widely talked about with practical efforts, there is still a distinction between the type of decisions made by the key user representatives. User representatives are said to be on site to "make business decisions quicker". Still the responsibility of design decisions mainly lies with the development team. In this connection, it is also relevant to note that even the arrangement where user representatives are temporarily collocated with technical team members mostly in the premises of the software house (i.e., isolated and away from users) for the duration of the project does not work (the experience with Organization C reported in Chapter Two is a case in point). There is a tendency among users that with such arrangement the user representatives become more and more part of the engineering (software) organization and less useful as users surrogates. Overtime, they become more empathic to the engineer's challenge and less connected to their previous work.

The contention here, therefore, is for fulltime and full membership assignment of selected users in the development team. This proposal demands users and technical experts to work together collaboratively: to understand a given business situation; to invent actions to improve the situation; to jointly and gradually discover and develop the software solution to support the business situation. To operationalize this, it is necessary to make provision in the contract for the creation of collaborative development teams composed of: technical members with knowledge of software development (software engineers, method experts, etc.), members with knowledge of the application domain (key user representatives, business process experts[13], etc.), and members with knowledge of project or change management (external expert). In such a set up, users are given the opportunity to fully and actively engage in the redesign of business processes and discussion and exploration of design options, over and above the usual passive participation. What is more, as recommended by Pomberger (2006), an external expert may be put in charge of the project for better performance, but with the amendments suggested by this researcher in Section 5.3.3 with regard to the role and mode of operation.

To make such a collaborative development strategy successful, however, it is important that from the very beginning:

- such arrangement is jointly negotiated and agreed upon in the contract;
- individual as well as group roles, responsibilities and compensations (incentives) are clearly defined;
- consensus is reached on project objectives and plans among collaborators; and
- a spirit of mutual support and teamwork is created.

These and related teamwork principles, concepts and techniques should be properly developed and nurtured, for the mixed professional grouping to create a special bond among the project team - esprit de corps. Building and maintaining esprit de corps takes a strong commitment and continuous effort by all parties involved and mostly by the project management.

Another important issue worth considering in connection to the productivity of such a collaborative team is motivation, particularly ways and means of improving the existing low motivation. The issue of people's motivation at work has been studied by many. In this connection, the classic/seminal studies which are widely cited in the literature and summarized by Hall and Fernandez-Ramil (2007:34), include:

- McGregor, in his Theory X and Theory Y, where he reasoned that motivation is deeply rooted in human biology and psychology;

[13] In selecting candidates at sub-team level, one is strongly advised to inlcude those users that actually do the tasks that the software is to support.

- Herzberg, in his motivation-hygiene theory, where he reasoned that seeking personal growth from work is the most motivating factor which in turn depends on the circumstances under which people work (work conditions, salary, interpersonal relationships, etc.);
- Maslow, in his hierarchy of needs, starting with the basic needs (such as the physiological needs of sleep and rest, food and drink, shelter, and air; safety from harm in a stable and secure environment, in that order) as foundations for the higher growth-related needs (including love and belonging, self-respect and respect of others, knowledge and aesthetics, and self-fulfillment, in that order);
- Hofstede, in his study of attitudes and values of people within an organization located worldwide, where he identified four cultural dimensions (power distance, individualism-collectivism, masculinity-femininity, uncertainty avoidance) that introduce great diversity across cultures when considering a more global view of motivation.

Beck (2004: 24) also talked about meeting such needs as basic safety, accomplishment, belonging, growth, intimacy, etc. for good developers.

As motivations vary between nations, organizations and generations, one needs to take these studies as starting points or guidelines to look into the issues further in the local context and from the view point of balancing the needs of the individual with that of the team.

In general, however, regardless of context the need for personal development and social contact is among the important sources of motivation. Accordingly, providing a stimulating social environment within which project teams can learn and grow through exposure to appropriate technology and best practices, and interaction with knowledgeable colleagues can all contribute to the motivation of team members. According to Hall and Fernandez-Ramil (2007:34), “people work because they enjoy it, requiring an adequate financial compensation to meet their ‘hygienic’ needs, but after that it is other, non-financial, factors that create motivation”. …“people are motivated by opportunities for self-development and social interaction and contribution to their organization and society".

To conclude this section, taken together, it may be difficult in reality to totally avoid cost overrun, time slippage and unsatisfied demands, particularly with large projects. However, emphasis on good project design (that addresses the issues discussed above) and the application of appropriate project planning and management continuously throughout the project will no doubt help in keeping the critical success factors within bounds.

In the local situation, in particular, lack of competence in project design and management happened to be one of the exceedingly prevalent causes for the failures. To this end, the introduction of project planning and management as one of the major activities in the software process, and the introduction of related techniques and tools in software methods, are a must in addressing software development in the local setting. With regard to techniques and tools, the possibility of adapting such industry standard methods as PMI's Project Management Body of Knowledge (PMBOK) and Prince-2 to software development projects is worth exploring. Practical experiences in this direction, both in teaching and work practices, indicate encouraging results. Samples of such customized project management tools used in course projects at AAU and CTIT and in real-life projects such as the third phase project of Organization B reported in Chapter Seven, are attached in Appendix 4 of this report.

5.4 Application Production

As introduced earlier, the Application Production process deals with the design and construction of the application software identified by the Project Design process. The detailed activities involved in this process are depicted in the diagram shown in Figure 7 below.

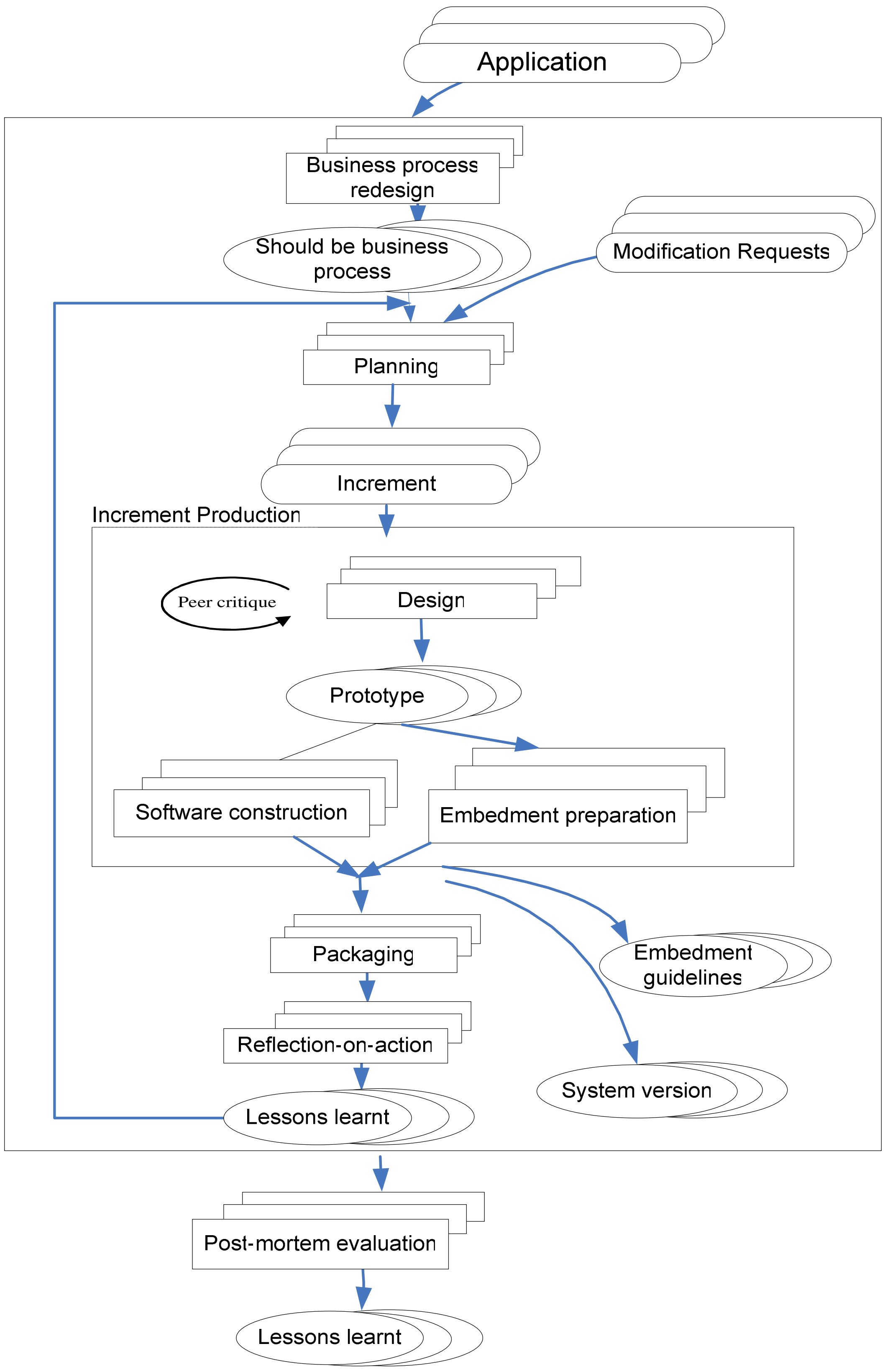


Figure 7: Reflective Steps - Application Production

Most of the activities in this process are dealt with at length in the original papers of the STEPS model (Floyd 1989, Floyd 1993) and in published software process literature (Boehm, 1988; Pomberger 2006; Cockburn, 2006). For this reason, the following discussion is deliberately limited to highlighting those activities and features salient to Reflective Steps. The business process redesign activity has already been dealt with at length in Chapter Four of this report.

5.4.1 Technical Planning

As shown in Figure 7, the application software production cycle actually starts with a planning activity. In a collaborative development environment, it is important that the team be very familiar with the software process to be followed in the development, to understand the business application area and generic features of the product required, and to be able to open and maintain a dialogue with team members and users. In collaborative development environment of Reflective Steps, planning involves these activities in the manner explained in the following paragraphs.

Although the basic assumption in almost all projects is that the team members assigned to develop the software already have the required knowledge on the application area for which the software is to be developed, in practice this is rarely the case. Software developers spend a considerable amount of time learning about the application area from key user representatives and domain experts. Within the traditional process model, this is usually done under the guise of requirements engineering or system analysis. In a way, in the process software engineers learn about the work practices and future needs by working with users like an apprentice through observation and asking questions.

Similarly, realizing participatory design through a collaborative team arrangement requires the user representatives that join the technical staff to be trained in basic software design concepts and procedures to the extent of enabling them to communicate effectively with the technical team in project matters. They need to be trained to appreciate the tradeoffs between requested functions, the capabilities of existing technology, the delivery schedule, and the cost. Terms and concepts need to be clarified to avoid misinterpretations at various levels that may lead to conflicts.

In short, success requires domain experts educating software experts about the application domain, and software experts educating domain experts about the software development process. Successful software development presupposes a thorough familiarity with the business of the local setting including the extraordinary non-standardized work practices, familiarity with new and emerging technologies and development tools, etc. As clearly demonstrated in Chapter Four, business process redesign activity provides a unique opportunity for building knowledge on the existing business practices. It is relevant to note that in the proposed redesign technique, it is users in Operation Group that lead the technical people through the process.

In addition, as part of the planning activity, more specific activities need to be introduced to familiarize members of the development team on the software approaches and technologies to be employed in the actual conduct of the software development work. The approach proposed for the project needs to be compared to the experiences of the project members and adjusted based on their feedback. In due consideration of the importance of such an activity, it ought to be separately accounted for in terms of effort, time and cost as part of the project estimates. For smooth and successful implementation, both the user organization and the software development service vendor should be willing and prepared to cover the cost of such activity. Needless to say, this would pay back by increasing the productivity of the collaborative team (Curtis et al., 1988).

As part of this planning activity, consideration needs to be given to exploit technological developments as specified by Pomberger (2006). Standard ideas in software design (well-proven experiences or best practices that deal with specific, recurring problems in the design and implementation of a software system) are usually passed on as tacit knowledge from experts to less experienced ones. As of recent, such experiences are being passed on (made accessible to others) explicitly in the form of patterns and reusable components. It is, therefore, important to learn about such design ideas and artifacts, as well as how to adapt or use them in the project during the planning exercise.

With regard to software methods and processes, selection and tailoring of starter methods should be done as part of the planning activities. Where there are existing institutional standards or formally documented methods or guidelines, these may form the starter methods. Where this is not the case, depending on the skills and exposure of professionals involved, one may choose to start by simply adopting suitable methods from the publicly available ones. Where all this is not feasible, the following approach proposed by Cockburn (2006), may be considered.

> Cockburn (2006: 149) defined methods as: "your 'methodology' is everything you regularly do to get your software out. It includes who you hire, what you hire them for, how they work together, what they produce, and how they share. It is the combined job description, procedures, and conventions of everyone on your team". In short, "a methodology is the conventions that your group agrees to". Whether properly documented or not, according to Cockburn , a method can serve several uses, including introducing new people to the process (how work is done in the project), delineating responsibilities, meeting the requirements of (giving confidence to) sponsors & demonstrating visible progress.

With this understanding, according to Cockburn (2006: 215), one may choose to start by simply writing the work conventions (based on the experiences of project staff and the company) as a list of sentences and then cluster these into topics. Among the possible topics to be included may be the following:

- Definition of common terminologies: for instance, people need to be able to understand what is meant by: project, iteration, increment, design, process model, method, technique, tool, etc.
- Description of documentations required as work products and their respective formats: project management plan (format and content); report content, format and frequency; modeling language, etc.
- Communication plan.

The assumption here is, although not articulated and documented, often practitioners and companies use some sort of in-house rules and procedures to keep track of progress and project staff performance, and mechanisms to coordinate activities within the development team and between the development team and user counterparts. All these and related input could be used in the process of drafting a method for starters. The document so drafted can then be enriched in a series of brainstorming and reflection workshops to produce the zero draft. Such a draft guideline is then continuously revised and enriched based on project experience as indicated in the next section.

Additionally, planning at this stage includes the definition of project sub-teams, tools and resources, risk assessment and controlling issues, etc. It is important that such plans are jointly developed by all concerned. Still review of the plan by the plan critic panel that may be composed of project stakeholders (sponsors, consultants, user representatives, etc.) may help.

Still, as part of the planning activity, high level design of the application system has to be developed. The architectural foundation of the application consisting of the increments that make up the application software and related work breakdown structure should also be drafted. As part of the architectural planning, each application software needs to be organized in manageable components (called increments in our case). Such an architecture is also expected to depict the relationships between the increments and include a starter sequence based for instance on business value, significance, etc. With regard to sequencing, a good strategy is to get started with the core business increment and proceed with those increments that expand functionality. During production, each increment may be further decomposed into modules or components. Assignment of corresponding sub-teams that deal with the development of the various modules and components (see below) required for each increment needs to be done as part of the planning well. From experience, as reported in Chapter Three, a sub-team composed of two software engineers and one business process expert has given good results.

Taken together, as with the other activities of the Reflective Steps approach, details of the planning activity need to be worked out with the full participation of members of the development team and based upon the knowledge obtained about the business application and technology platform to be used. What is more, as clearly shown on the diagram in Figure 7, this needs to be regularly reviewed and updated based on knowledge gained from project experience. To reiterate, it is not the plans that are important but what is obtained in the planning process.

5.4.2 Increment Production

In Reflective Steps, the development of each application software involves the delivery of a number of increments. In other words, development of an application is to be carried out in steps, where each step ends in the production of an increment, and its integration/packaging into an operational version. After increment packaging, there will be a period of reflection set aside to: assess progress, process and context; to explore their implications and write up lessons learnt, etc. From this angle, an increment is a piece of operational software that is ready for packaging with the increments already (previously) delivered to users. As such increment production concerns the actual production of a software component that supports a coherent task of the business application. The reflection, on the other hand, is a collaborative communication and learning mechanism that provides for better understanding of requirements, design options and applicable methods, based on the feedback from concrete experience and experimentation. Among the possible outcomes of such reflection exercises are revisions of plans and processes to enhance the quality and timeliness of subsequent software increments. Such incremental mode of development is being considered as a critical success factor in developing software (Cockburn 2006).

Typically, each increment development starts with updating the development plan. As part of this process, to introduce better management and flexibility, each increment may further be decomposed into modules, and each module into components. It is, therefore, important that the increment architecture (which is an element in the application architecture) be worked out first to facilitate smooth and flexible increment development. Any of the available object-oriented design techniques may serve this purpose best. In a related work, attempts are being made to explore the application of the 'function-mechanism' framework, proposed elsewhere (Miyake, 1986) to study how people try to understand complex things, to supplement the object-oriented techniques in decomposing increments into modules and components.

In general, the design process at each level involves multiple cycles of design-critique-reflection based on prototypes and reflection workshops. Typically in each cycle, one group presents the design, while the other asks questions to understand the design and design choices, followed by a joint session where the group as a whole engages in resolving the issues raised during the discussions (see Section 6.3.4 on RS Workshops). The domain of discourse in each cycle involves: understanding requirements (both actual and potential), visualizing design options, collective design critique and feedback (reflection-in-action), programming and testing designs. In particular, supported by continuous communication and learning, the following design activities are carried out for each increment until consensus is reached on an acceptable product:

- requirements understanding and scoping: identifying the components and boundaries of the application to be developed, understanding the existing and planned processes and requirements;
- envisioning[14]: visualization of the newly required system (that meets both current and future requirements); and
- realization (designing a system to meet the requirements): this may involve the use of scenarios and use cases, prototypes, etc. to surface issues, explore alternatives, test assumptions, etc. and the construction of the actual software based on the understanding reached.

In the cyclic process of design-critique-reflection, the outcome from a cycle leads to refinement and adaptation in each of the domain of discourse in the subsequent cycle.

In RS, the workshops on technical design issues are to be supported by the use of prototypes. Needs less to say, prototyping is a firmly established technique in software

[14] The term envision is usually used in the sense of setting a shared goal (as strategic planning) in organizations. In the context of system development, a vision appears as an initiating idea that defines new direction, create a new thing, or bring about a new way of doing business (Jones, 1997).

engineering, particularly when it comes to testing design concepts and clarifying requirements understanding. During design it can be used to develop screens, dialogs, transaction process and updates, report and inquiry formats; and it can be used for feature enhancements during integration and use. Here the use of operational prototypes and joint programming and testing with users (in particular, two software engineers working with a domain expert) are both critical and recommended as a vehicle for successful development. Such operational prototypes also provide: an efficient means of capturing and communicating requirements, a facility to demystify technical issues and artifacts in the discussion with users, a better means of engaging users to codesign the software and ensuring its acceptance etc. In the process, such operational prototypes are continuously enhanced until eventually they become the final software product.

An increment may require several iterations before it reaches an acceptable level, where each iteration brings more adaptation and refinement to meet user expectations and preferences. Once the increment comes out of the process successfully, and meets the minimum requirements as specified in the test results, the increment is said to be ready for deployment. At this point, an executable version of the increment, together with guidelines to package the increment with already released versions is delivered. In case there are previous increments released and in operation, the last week of the increment is better used for preparing the increment for delivery together with necessary interfaces and related requirements to integrate/package the increment with earlier ones. For this purpose, the increment is passed over to the packaging group who will put this together with other earlier increments into an updated working version. Such integration also helps to ensure integrity across all increments that have already been packaged into the working version. Errors detected in the integration are corrected as part of this process. As required, refactoring of some of the previously delivered modules may also be done. The deliverable from such packaging exercise is a revised version of the already installed operational software. Based on the redesigned business process and the revised version, guidelines necessary for work practice revisions are also updated.

As explained in Chapter Four, a critique panel (representatives of experts in the business process, technologists, service receivers, etc.) is used to critique the process design proposals from the operation group during business process re-design. Similarly, in the process of integration, a use panel which is composed of the operation group (and some more actual users - those that directly operate the software in the course of discharging their duties and responsibilities) may be used to give feedback on the newly integrated version of the software before it goes live. Such a panel should be given some time to go through the newly integrated version. This is followed by a reflective workshop to discuss and review the product based on the feedback from the use panel. The production team will then revise the product based on the resolution of the workshop.

An increment cycle ends with an assessment workshop (reflection-on-action) on the increment that aims at improving the software process and the product in subsequent increment cycles. Reflection in general constitutes the ability to uncover and make explicit to oneself what one has planned, observed, or achieved in practice (Raelin, 1997). In addition to flushing out what is working well in the process, product, schedule, organization, etc., it involves brainstorming fixes for the problem identified. With respect to each increment, reflections take place at two stages: within the increment and at the end of the increment. As discussed earlier, the reflection within the increment is more of a group-level reflection-in-action (between the group that came up with a design to get immediate feedback and others that serve as a critique panel, similar to the reflection between a tutor and a student). It aims at experimenting with the new ideas right from the increment. According to Raelin (1997), reflection within the increment could also help in bringing about the inherent tacit knowledge of experience to surface and reconstruction of meaning. The reflection at the end is a more critical reflection at project team level to learn from the concrete experience of developing the increment (reflection-on-action), and revise or refine plans and the way of work for subsequent increments (reflection-for-action). Such reflection will give an opportunity to evaluate what was already done and the way it was done based on the feedback from real-life experience. Accordingly, in such reflection sessions mistakes must not be hidden but need to be exposed and learnt from (Beck, 2004:29). Lessons learned are used to streamline workflow and change things that did not work properly, invent new ways of working, reorganize team structure, reallocate resources, get more training, revise increment architecture, etc. as required. In short, this provides an opportunity to revise plans and improve development processes to be used in subsequent increment developments. It also provides an opportunity for resolving risks and critical process and design decisions early rather than late.

For the last increment of the application package, the assessment is done as part of the application software postmortem assessment workshop. The reflection at this last stage tries to address higher level project issues more critically. Based on real-time project experience, it tries to particularly look into problems that have occurred within the work setting to perform within quality, cost and schedule goals. It may also be used to question the competence of the development team setup, the priority and scope of projects and applications, the business processes in place, etc. and take related corrective actions. Reflection at this level is reported to the management or alternatively could be jointly made with management at Project Design level. The operation of putting in place measures to address weaknesses in processes which have been identified as sources of defects or risks to quality, cost or schedule performance will also be jointly agreed upon.

While it is left up to each group to organize technical or informal meetings every day or every other day such as those recommended in agile methods (Cockburn, 2006), there should be regular weekly meetings to collectively review progress/status, critique design options and reflect on process and work habits (for more on this see Section 6.3.4 beyond).

In terms of schedule, although this may vary based on the size and complexity of the application systems, an increment is better planned for delivery in a fixed period of time. Such time boxing is necessary not to let the iterations go on for ever. In this connection, it is relevant to note that unnecessarily shorter periods make it difficult to get enough feedback to produce a usable product. In the experience so far, a period not more than six weeks seems to be reasonably adequate to deliver a piece of software increment. In case the size of an increment is large and involves many modules, among the strategies to be considered to keep the time constraint within bounds is to deploy more resources so that different groups work on different modules in parallel or introduce overlapped development.

The reflection scheme proposed here seems to be somewhat similar to those mid- and post-increment reflection workshops suggested by Cockburn (2006). One difference lies in the timing of reflection within increments. While Cockburn suggests daily [informal] reflection, and one mid-increment reflection, the suggestion here is to have: design reflections routinely on a weekly basis, assessment-like reflections at the end of each increment delivery and at the end of each application delivery. Another difference lies in the levels at which the reflections are made. The suggestion in Reflective Steps is to do the reflection within an increment at work group level, and the post-increment reflection at project level. No distinction seems to be made in Cockburn's suggestion. Yet another difference is, Cockburn suggests a somewhat oral and informal mode of reflection supported by flipcharts. The suggestion here is to introduce more critical reflection supported by formal documentation during and following the reflection sessions in the form of reflection journals. For this purpose, the provision of the necessary support infrastructure to ensure the documentation, circulation and implementation of the lessons learned from the reflection is suggested in Reflective Steps. See next chapter for more on the details of the reflection modes and the support structure suggested, as well as the advantages of maintaining a reflection journal.

Before concluding this subsection, in view of the controversies around the subject, the following note on the need for reflection on **written documentation** requirements at various stages of the production cycle is felt in order.

As much as too much emphasis on strict adherence to defined documentation is discouraged, the author feels its omission altogether is counterproductive. In a project environment highly characterized by high turnover of project staff and where there is lack of historical data and organizational memory, the importance of written documentation, over and above the comments in source code, cannot be overemphasized.

Written documentation is important and useful for a number of reasons. It is an important instrument to create a common and shared experience particularly where the community consists of inexperienced members and members with varied professional background. It helps new members to relatively quickly familiarize themselves with the way things are done in the project. It also helps to track and coordinate work among the various parties involved in the project. Written documentation is very important not only to successfully complete and deliver the software at hand but also for subsequent projects that relate to maintenance and upgrading of the software. Successfully completed projects that do not have written documentation are bound to run into problems at the time of maintenance and upgrading particularly in cases where the people involved in the development are no more with the projects. This is the situation that features most in the local environment as reported in Chapter Two and Chapter Three.

On the importance of documentation, even agile approaches (or Agilists), who initially are frequently cited in their insistence to discourage documentation and formality, stated

> "Ideally, documentation activities are deferred as long as possible and then made as small as possible. Excessive documentation done too early delays the delivery of the software. If, however, too little documentation is done too late, the person who knows something needed for the next project has already vanished." (Cockburn 2006:219)

And by way of recommendation, Cockburn (2006:221) wrote:

> "Bear in mind that there will be other people coming after this design team, people who will, indeed, need more design documentation"

On the other hand, written documentation and modeling are also criticized for being time consuming and bureaucratic and less readable by designers. With some methods, a lot of time is spent on producing too much documentation, particularly from design modeling and requirement engineering phases to the extent of risking the successful completion of the project. Based on his discussions with successful project teams, Cockburn (2006:36) reported on what the teams told him about the limitation of modeling in software development:

> "the interesting part of what we want to express doesn't get captured in those models. The interesting part is what we say to each other while drawing on the

> board [while modeling] …. We don't have time to create fancy or complete models. Often, we don't have time to create models at all"

As such, religiously following documentation on model building methods and spending a great deal of effort on developing fancy documentation and models, may significantly affect the delivery of the software. The purpose of the software project is to deliver usable software, not constructing models and documents. In this sense, the documentation and models are means to an end (the software product) and not ends by themselves. Accordingly, it only makes sense to construct models and documents to the extent that they provide sufficient information to the recipients so that the recipient is enabled or positioned to take the next move (proceed with subsequent steps) of software construction.

In Reflective Steps, the decisions related to how much, where and when to model and document are determined more through the collaborative reflection session held at the various stages of the project. We strongly advocate, however, the maintenance of reflective journal in the manner defined in RS Workshop (see Section 6.3.4 beyond) as a documentation that really adds value for both the software development and process improvement projects.

5.5 Application Use

The need to consider users' views in the course of designing a usable software system was repeatedly emphasized by various workers.

> "The reason people design difficult things is their lack of empathy with the users. A good designer has to take the user's point of view, consider what information the typical user must have to work properly, easily and efficiently. The best way to do this is to observe and interact with the typical users, testing out the designs on them – and being willing to change them when users find themselves confused. Alas, most designers concentrate on cost, or aesthetics, or technical details." (Norman,1993).

According to Norman (1993), a good writer and a good designer have to share certain characteristics. To be successful, they need to understand the needs and abilities of their audience, and they must consider just how their product will be used. Their designs must be tested, tried out, and then revised. As writers are tested by editors and readers, designs need to be tested by colleagues and users. Things that are unintelligible or even dangerous to the average user are likely to seem perfectly reasonable to the designer. Thus, feedback about the state of a system is essential in normal social intercourse.

In RS, in addition to the design, construction and testing of the software during increment production, the development continues to the use phase. As documented in the cases reported in Chapter Two and the experiences of practitioners reported in Chapter Three, users and developers immerse themselves in the actual software design when they perform their real-life tasks with the version of the software installed for actual use. Once the installed versions are up and running, the development proceeds with adding to and changing/adjusting the software to better support (and fit in) the real work practices based on the mistakes and problems uncovered by both developers and users while in use. In short, the use process enables users to provide timely feedback based on actual use, enables developers to fix problems and track change requests.

Usage addresses the process of handling these and related issues of designing a system for its context of use, by working closely and collaboratively with users. A good account of the surrogate activities and related issues of the Application Use are well documented in the writings of Floyd et al. (1989).

Mitchell Kapor (quoted in Jones, 1997), expressed:

> "the lack of usability of software and the poor design of programs are the secret shame of the industry. Given a choice, no one would want it to be this way. What is to be done? Computing professionals themselves should take responsibility for creating a positive user experience. Perhaps the most important conceptual move to be taken is to recognize the critical role of design, as a counterpart to programming, in the creation of computer artifacts …"

In this connection, it is relevant to reiterate the warning expressed earlier with regard to overdependence on users. This, however, need not be confused with the spirit and assumptions underlying the use design. As warned earlier, developers need to be cautious of certain design traps that may negatively affect product quality. The tendency by developers to design and build whatever the users say, even if they are sure that the ideas are silly, is counterproductive. Sometimes it may be appropriate to opt for a design solution that challenges the users' habits but better supports their requirement. In working with users, it is also relevant to note that sometimes users are either not quite sure of what they want or may continuously put forward unsettling and conflicting requirements. Likewise, the development team may also exploit the opportunity for an excuse of escaping the pressure and challenge from the situation by shifting the burden over to the users.

Beyond facilitating the design of a usable system through active user participation and collaboration, Application Use ensures that software versions delivered to users are smoothly deployed and used. The detailed activities involved in this process are depicted in the diagram shown in Figure 8 below. A more elaborate account on most of these activities may be found in Floyd et al. (1989) and Floyd (1993).

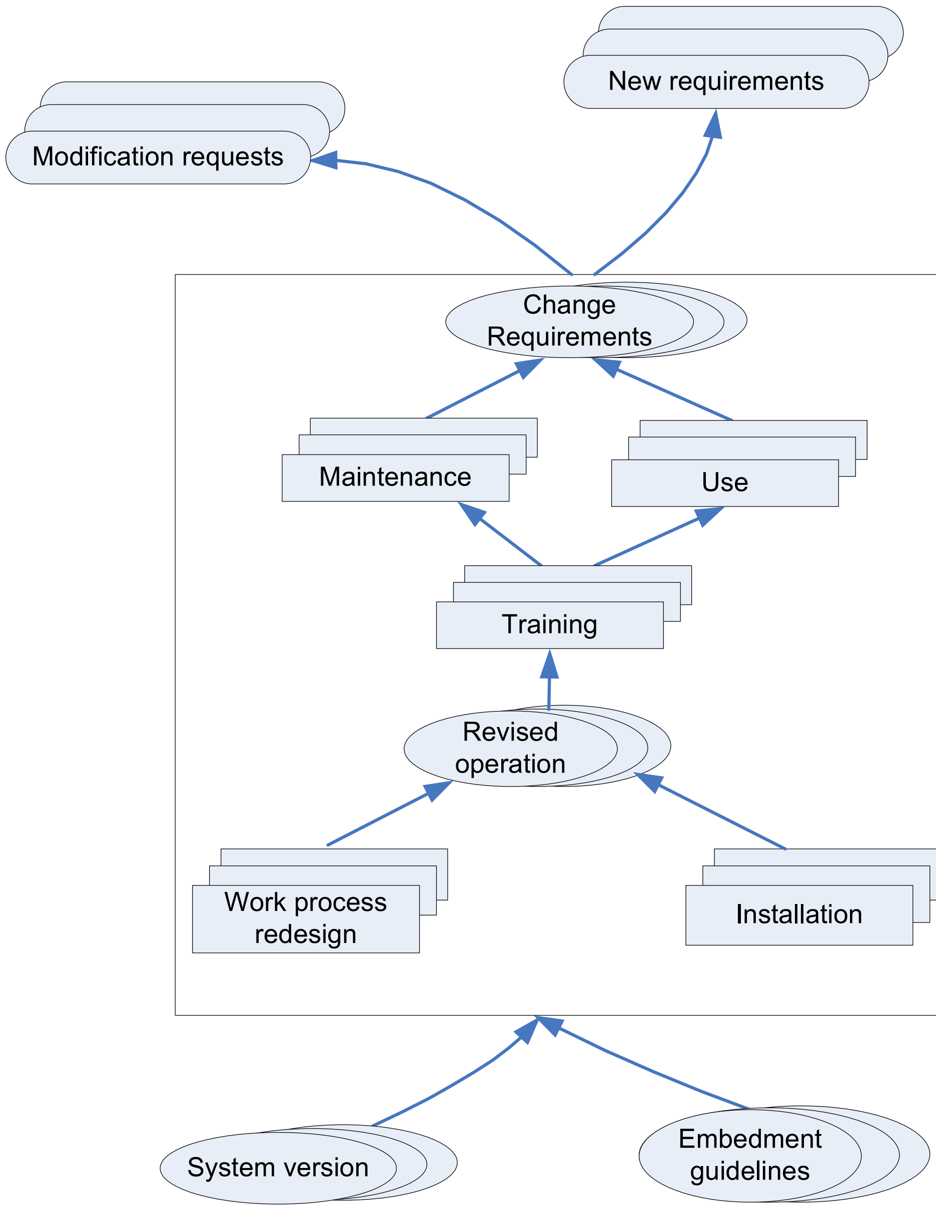


Figure 8: Reflective Steps - Application Use

The introduction of the system, if it is large or sensitive, may create turbulence in the client organization. To address this, the process needs to be accompanied by appropriate change management efforts at organizational level. At project level, such change management related issues as training and work process redesign need to be considered as part of the Application Use. In other words, for the organization to benefit from the software developed, just installing and enabling the software alone is not enough. The people who will use the software need to be enabled as well. This includes end users of the software, as well as technical experts, who are responsible for the use and maintenance of the system, respectively. With the successful implementation of the newly installed software programs fully integrated with business process redesign there will come a series of changes. The way the business is conducted including the role of individuals in the process will change. Accordingly, the skills required by the operatives will change. For this reason, it is important that we foresee some of these inevitable series of changes and devise appropriate training programs at various levels: managers, operatives and technical support. For the purpose of facilitating such training programs and also for smooth implementation of the redesigned processes, related operation procedures and manuals need to be updated based on the features and capabilities of the software developed and related embedment guidelines.

In this connection, it is also relevant to note that full integration of the system in the organization will be gradual since it is difficult to change the way people are used to doing things at once. While the business rules and procedures may be changed overnight, the norms usually change only gradually and through the interaction of people at all levels in the organization. To this end, targeted awareness and training programs must be continuously conducted. Such training programs should also provide forums where the people involved can discuss and negotiate the various issues related to the changes, and make these changes part and parcel of their daily activities. Such forums are important to fully integrate the redesigned business process and related software into everyday work routines.

Similarly, appropriate maintenance and support strategies need to be in place to promptly address concerns and requirements resulting from the use of the software and related discussions and negotiations. Software systems introduced into an organization (those that are used by the people as integral parts of their daily activities – used not because they are told to do so but because they find them useful) inhabit within and are used inside the environment while at the same time affecting the environment. Once such systems are introduced, the systems are likely to co-evolve with the organization resulting in the need for maintenance, upgrading and development next versions of the systems.

A need for change is identified when users experience difficulty using the software to process certain transactions or retrieve certain information to support decisions, etc. If such difficulty comes from lack of training or misunderstanding, it may be overcome through the provision of appropriate training, guidance and support. If, however, it comes from software deficiency and if such deficiency is minor, the software technical support is called upon to get it fixed. On the other hand, when addressing the deficiency requires more time and possibly redesign of the software, a request is filed for a revised version of the software. In the mean time, manual methods such as the introduction of work-arounds, recording the required information with personal notes and records outside the system may be used. In this connection, it is also relevant to note that with more user involvement in the development process and the opportunity to witness the capabilities of the system from experience with the installed increments, users often tend to ask for more features and capabilities which may result in new requirements. Where such requirements are not minor, they need to be jointly negotiated before incorporation. As required, they may also be escalated to the management for decision where resource questions are involved. These and related factors are addressed in Reflective Steps by the various activities incorporated within Application Use and related communication between Application Use and the other components as clearly indicated in the diagram shown in Figure 8.

5.6 Post-mortem Assessment

An integrated version of Reflective Steps that synthesizes the various components discussed above, is presented in the diagram shown in Figure 9 below. For consistency, a version of the Project Design component (previously depicted in Figure 5) is reconstructed in Figure 10 using the diagramming conventions of STEPS.

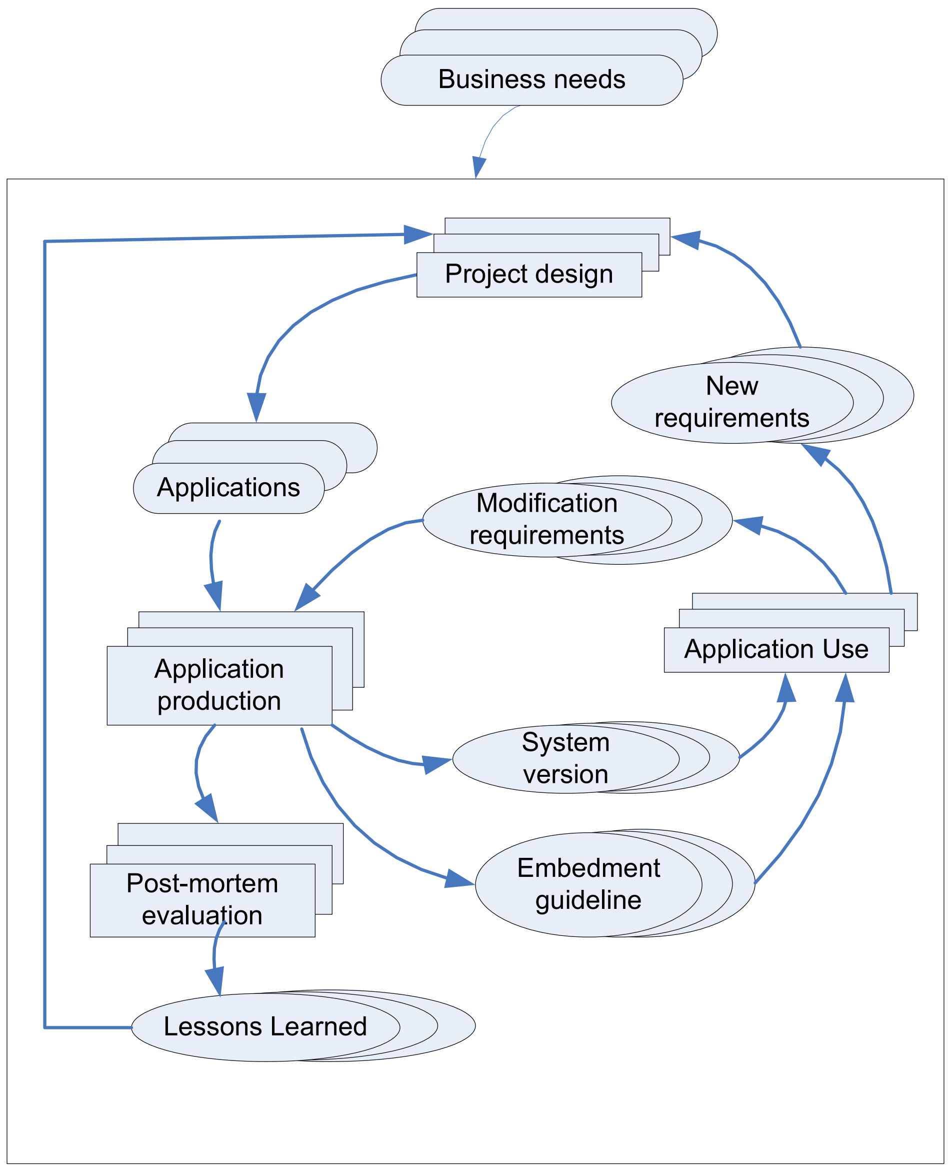


Figure 9: Reflective Steps – Overview of the Integrated Model

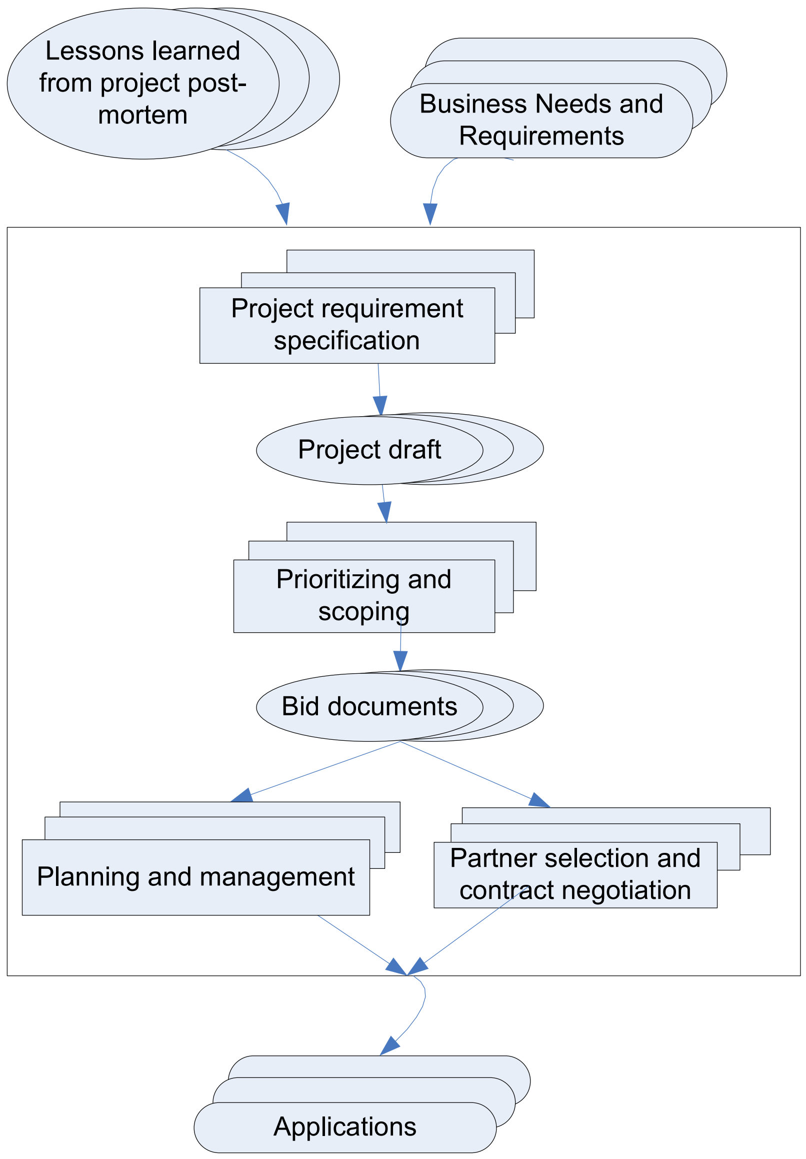

Lessons learned from project post-mortem
Business Needs and Requirements
Project requirement specification
Project draft
Prioritizing and scoping
Bid documents
Planning and management
Partner selection and contract negotiation
Applications

Figure 10: Reflective Steps – Overview of Project Design

As can be seen in the integrated version, the last activity after the development and delivery of an application within a project is a post-mortem evaluation. A post-mortem evaluation is an activity which is part of the current project but one that aims at forming the basis for the next version. Particularly, in the type of software development under reference here (i.e., custom development of an application software for an organization), the development does not end on delivering the software. Among the project close-out activities is a post-mortem evaluation of the project. This will serve to provide an advantageous position for what Cockburn (2006) referred to as the second major goal.

The first goal relates to delivering the software. The second goal relates either to altering or replacing the system or creating a neighbouring system (in our case this equally applies to the next application development project).

The usual question in this connection is when to do this post-mortem. Two commonly considered and practiced options are: to do this at the various intermediate stages of the project along with other intermediate work products, or to do it at the very end of the project. Neither of the options is optimal on the grounds that the former introduces considerable extra work and cost (to the extent of jeopardizing the timely delivery of the software), while the latter would run the risk of under-documentation or no documentation at all. Some workers (Cockburn, 2006) have attempted to provide a sort of guideline or advice on finding the balance between these two options.

In Reflective Steps, we propose an alternative way of finding a balance between the two options. In particular, the approach proposed relates to doing it at two stages. First, the capturing of relevant data for the post-mortem is to be made routinely by embedding this agenda in the routine project review/feedback and reflection activities. Where as the analysis and summarization aspect of the post-mortem is done once at the end of each application delivery.

The post-mortem report is compiled in the form of lessons learned for consideration and use as part of the project portfolio management at Project Design level and as part of the planning activity at the Application Production level. For these purposes, as part of the post-mortem, a guideline or implementation plan may also be prepared for use in revising the software process and plans on the basis of the results from the project together with suggestions on its institutionalization in the organization. Such plan may include detailed description of the revisions requested on the product, the revised process model, activities and their sequencing, roles and responsibilities of the main actors to be involved, the main steps needed to make the institutionalization process complete, time schedule, costs, and an organization for creating an infrastructure for supporting the whole process.

This concludes the introduction of Reflective Steps and its application and use for software development. The next chapter will focus more on the application and use of Reflective Steps for software process improvement to be carried out within, as part of a software development project.

CHAPTER SIX

6. Reflective Steps for Process Improvement

In Chapter Five, we presented how the proposed model could be used to support software development. In this chapter, we elaborate more on how the proposed approach may be used for software process improvement in the context of a typical software development project. First, justification for a project-based approach to software process improvement is presented based on review of related literature. This is followed by a discussion on learning in work contexts based on related literature review. With these two sections as a background, the third section presents the proposed approach for learning and process improvement in Reflective Steps. In other words, we will show how the learning within and between iterations at increment, application and project levels can be methodically used to improve the development process itself.

6.1 Project-based Process Improvement

Software process improvement, taken literally, implies the existence of a process to be improved. The findings reported in Chapter Three have indicated that most software development companies in Ethiopia did not have anything (at least for the moment) that can be recognized as software process. Yet in order to meet the existing and upcoming customer expectations, to be productive and stay competitive, the companies expressed strong desire to address this issue without further loss of time.

On the one hand, in the absence of processes, it seems that it only makes sense to first talk about the process of process development rather than process improvement. This notwithstanding, developing a process for the first time by itself can be considered as an improvement – in the sense of the transition from not having a process to having one. In reality, processes in small and new companies emerge from experiences of working on projects. Initially the processes exist in the heads of the professionals that have come together to create or work for the company. Such processes which are stored in the form of skill or tacit knowledge get the chance to be actually expressed when working on projects. The strategy adopted here is, therefore, to exploit the opportunity provided by a project to capture such knowledge for use in the same and subsequent projects. In this way, the projects can support the process of process development, thereby improving the situation from having no process to having a starter process. Once defined, the starter process could be continuously adjusted and refined based on the experiences gained during its use. At the end of a project, experiences obtained from the use and adjustment of the processes during the project period are further analyzed to update the starter process for consideration in subsequent projects. Such is how processes are to be established and improved in practice particularly in environments where documented processes are missing.

Commonly suggested techniques (Pourkomeylian, 2002) in the industry with regard to process improvement (SEI's CMM, ISO/IEC standard 9000:2000, Experience Factory Approach, GCM, etc.) are not only suited for grown up software engineering environments but are also expensive and heavy on process. The relative merits and deficiencies these software process improvement paradigms and their relative applicability to various organizational cultures are discussed elsewhere (Zucconi, 1995). The assumption by the popular process assessment and improvement methods that there is capacity to do this kind of work regardless of the size of the company and the situation they operate under and the culture local to the companies has been criticized (Nielsen & Pries-Heje, 2002). What cannot be disputed is that most software companies in developing countries, particularly those that operate in immature software engineering environments, do not have technical capacity or deep pockets that can afford the above improvement methods commonly recommended by the industry. Under the circumstances, therefore, there is a pressing need to come up with software improvement methods which are appropriate for these environments.

One of the strategies being put forward for developing software process improvement involves engaging independent process improvement specialists or experts to do an assessment of the existing situation and suggest improvement recommendations. Such strategies even go to the extent of suggesting the introduction of a full-fledged unit within the organization for this purpose, and in certain cases as a major requirement for the process improvement work to be successful. The extent to which such approach of engaging people that are not actively and directly involved in the software development work, or that have somehow been detached from the day-to-day routine of software development, would succeed in bringing about actual change in the process in a sustained manner is yet to be proved. What is more, so far, there does not seem to be adequate empirical evidence to substantiate this claim, while historical experience in similar but related undertakings indicates otherwise.

> This strategy seems to be identical to the approach taken in traditional Organization and Methods departments that tried to do similar things in the area of business process and work method improvements. The case of engaging systems analysts to go between users and software developers for the purpose of preparing design and test specifications on behalf of the main actors is also another example. In today's realty, such lines of work do not seem to have places, at least in their traditional forms. Instead of organizational units fully charged with such responsibilities, the activities are taken over by the main actors working jointly with consultants or external experts.

As with software development, where full participation of users is encouraged in the development of usable software, by far the approach that would be more effective is to make necessary arrangement for the users of the software development processes (i.e., developers) themselves have more say in the process design or improvement (Arent, 2000). That is, the process should be participative – based on the premise that the knowledge and resources necessary to improve the situation are distributed among a group of individuals involved in the process. The active involvement of the main actors themselves is likely to ensure that both the process design and improvement are grounded in the needs of the professional practitioners in the project. What is more, the practitioners who have participated in the process feel that the process and the improvement proposed are based on (driven by) their needs. By involving practitioners in identifying and improving their own problems, the improvements become situated in the proper context or practices (that is, in their daily activities). This will make it far more likely that the practitioners will be committed to change their practice. Needless to say, it is such feelings and commitment that create a sense of ownership of the process and better results by the practitioners involved in the project work. As required, however, external software process experts may be included to provide further support and guidance.

In this connection, Dahlbom and Mathiassen (1993: 55) wrote:

> " …to improve the way work is organized in a system development group, we cannot rely only on the abstract system that is expressed in the standard project model used by the group. For one thing, we should compare this system to the beliefs and attitudes of the project members and learn from the differences between the ideal world of the project model and the experiences and ideals of the projects. But more importantly, we should formulate alternative systems, expressing the perspectives of the involved actors on issues such as project organization, communication and cooperation, programming, testing, contractual arrangements, and project management. We should develop project models based on such systems, and compare them with the actors' views on present and future practices."

Taken together, both methods related to project management and software process need to be evaluated based on feedback from the application of same in real-life projects, instead of an independent study conducted for this purpose by specialists not directly engaged in software development project. They need to be updated/improved (based on experimentation in a real-life project environment) on a regular bases. The process improvement process can in this way benefit from project experience. What is more, process improvement is not a one-shot effort. It is an evolutionary, iterative improvement initiative involving continuous learning. Accordingly, a project-based bottom-up approach, strongly grounded in experience-based learning, to process improvement provides the opportunity to revisit and improve the process with each new project, thereby providing a continuous process improvement mechanism.

In this way, each software development project provides a unique opportunity for accumulating experience, knowledge and related learning which in turn is valuable for improving both the performance of projects and improvement of processes. In the local situation in particular, and in the case where there are no documented software processes, it is argued that better results may be obtained in terms of improving the process if such initiative is linked (integrated) to project management activities of a software development project. To this end, the recommendation is to embed process improvement as part of every software development project. For the purpose of facilitating this, in large projects, a process expert role may be introduced during project design.

> To avoid unnecessary workload on the project manager, in large projects a new role may be defined within the project management group for keeping track of process improvement related discussions, reflections and resolutions.

The process related reflections made at the various stages of the development process, and the learning thereof, could be used to support the improvement work. The lessons learned from the reflections are used to discover new areas for improvement, to implement incremental improvements to the project's practices, to monitor improvement progress and to provide feedback to project design. The process improvement aspect of the approach proposed inhere is more concerned with identifying the need for process change and trigger improvement initiatives during the process as a feedback. The same experience is recorded in the project journal for later consideration in process establishment (for those without a process already) or process enhancement (where there already is one), whatever the case may be.

As already explained in Section 5.4.2, the improvement process may start with the project-specific process elements (starter process) defined at the beginning of the plan. Included in such a plan could also be a generic guideline on how to continuously improve such a starter methodology in parallel with other project activities. During project execution, such a guideline may be followed to continuously assess through tracking and reflection (with the full and active participation of the project staff) the extent to which the process being followed is contributing to the achievement of project outcome. A continuous learning process should follow based on reflections on the work done so far and achievement of goals – particularly on the extent to which the process has contributed to productivity, quality and adherence to schedule.

Based on lessons drawn from the assessment results, in addition to other corrective measures such as: training on the use and application of the process, emphasizing or deemphasizing the extent of use of the processes, may be taken. In addition, the processes may need to be continuously adjusted in a manner that better contributes to the project outcome. It is also important to document and share the adjustments made, together with discussions that have led to the adjustments, the options considered, etc.

The approach discussed above is similar to the dynamic method tailoring commonly practiced in agile systems.

> "dynamic process tailoring, especially during and within the ongoing software development projects, has been highly valued in the principles of agile software development. The agile principle of regular team reflections of software developers in order to become more effective relates directly to the continuous and dynamic project-specific tailoring activity, whereby the organizational base process is iteratively tailored throughout the project by the software development team"(Salo, 2006: 46).

The principles of agile software development focus on iterative adaptation and improvement of the activities of individual software development teams to increase effectiveness. In this study, an attempt is made to extend such iterative adaptation and improvement at team level to project and organization levels. In particular, an experience-based learning approach strongly grounded on collaborative learning is introduced as key process to support multi-level process improvement. The extended process iteratively provides improvement aspects for immediate use in the same project. It also supports the evolvement of the organizational level improvements in the long run through a double-loop learning (see below).

6.2 Learning in Work Contexts

In a typical work environment, individuals and organizations learn from a process of feedback and information exchange through internal dialogue. As opposed to the practice in academic settings, where learning begins with knowledge and is then put into action, in work-based learning action or experience comes first and learning follows. In this mode, learning takes place where practitioners work in groups, discuss problems, and exploit the opportunity to learn from their own and others' experience. In such environments, for instance, whenever practitioners are faced with unexpected requirements in practice, they tend to reflect in action, consult colleagues and devise workarounds to handle the situation. Practitioners readily learn to accept and to discharge their real-life responsibilities by contrived exchanges with others (peers, users, collaborators, etc.) during the prosecution of real-life activities. They learn both to give and to accept from others the criticisms, advices and support needful to develop their own position, all in the course of identifying and treating their own personal tasks. Working in this way, practitioners can find practical solutions and learn collaboratively by combining real situations with theoretical knowledge.

(i) Communication and Learning as Key Instruments

The above learning practice is more or less commonplace in real-life software development work. Software development projects bring together collaborators with various backgrounds, skill sets, organizational culture and experience levels (software engineers and process experts from within the user organization and the software development house, key user representatives and domain experts, managers, consultants, etc.). This brings a rich tapestry of backgrounds, valuable knowledge and experience, organizational contexts, roles and aspirations into the project environment. As much as the situation provides opportunities for information exchange, experience sharing and learning, it may also introduce communication problems leading to significant disagreement among members of the project team. In the effort to maximize benefits from the opportunities provided and minimize the problems to be encountered, the role of communication and learning instruments were repeatedly emphasized (Floyd et al., 1989;

Mathiassen and Nielsen, 1990; Pomberger, 2007). As of recent, such emerging methods as agile, on the basis of the understanding that software development is a game of invention and communication (Cockburn, 2006), have started to emphasize the communication aspect more, together with suggestions for applicable modes of communication. However, not much seems to be done about the learning aspect, particularly in terms of providing practical techniques.

The availability and use of learning and communication instruments becomes more critical in the local environment, where there are younger talented but less experienced engineers, and where there is high staff turn over resulting in (the continuous drafting of) new member at various stages of the project. In an attempt to address this gap, Reflective Steps incorporates a learning model that has its basis on established theories and practices in the area of work-based learning.

As discussed in Chapter Five, in the software development practice we are more interested in such types of learning elements as peer-feedback-based action learning that encourages cycles of design-critique-reflection iteratively. accordingly, the learning incorporated is conceptualized as a cyclic process that involves in each cycle,

- taking action based on prior experience and a plan enriched by critique,
- critically reflecting on the outcomes of the action,
- drawing lessons from the reflection,
- taking action based on the learning, etc.

(ii) Theoretical Perspectives

Reflection is already established to be an important part of the learning process and there are many theories about what reflection is and why it is so important especially for learning from experience, developing the skills of professional practice and for the development of meta-cognitive skills which are said to enhance learning.

> According to Dewey (1933) and Moon(2000), we don't learn from experience; we learn from reflecting on experience. Learning “involves creating new insights and integrating them into your existing awareness and knowledge” (Taylor, 2004:78)
>
> We reflect in order to learn something, or we learn as a result of reflecting. Reflection plays a mediating role by transforming meaningful experiences into learning (Vygotsky, 1978; Kolb, 1984)

The type of learning under reference has its roots in the following learning theories:

- deutero-learning or learning to learn by Bateson (2000) that outlines moving from habitual response to learning from context and stepping outside;
- organisational learning by Argyris & Schön (1978) that deals with the development of a theory of action learning; and
- experience-based learning cycle by Kolb (1984) that addresses learning as a cycle of experience, reflection, generalization, and testing.

According to Sørensen (1999) and Raelin (2001), for Bateson, the key to understanding the learning process were the phenomena of change, context, and the recognition of ‘context of contexts’. For Bateson, learning denotes change of some kind. Considering motion as the simplest and most familiar form of change (described in terms of “position or zero motion”, “constant velocity”, “accelerate velocity”, “acceleration”, “rate of change of acceleration”, and so on), Bateson delineates a set of five classes of learning labeled Learning 0 through 4. As depicted in Figure 11 below, Learning 4 encompasses Learning 3, while Learning 3 encompasses Learning 2, Learning 2 encompasses Learning 1, and Learning 1 encompasses Learning 0.

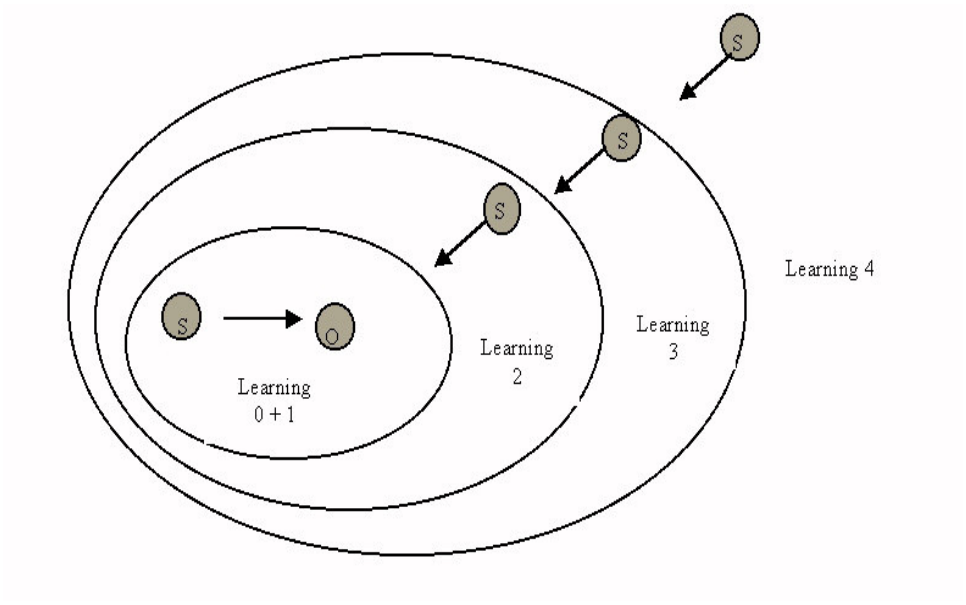


Figure 11: Bateson's view of the Learning Phenomenon (adopted from Sørensen, 1999)

According to Bateson(1979), there is a developmental hierarchy to learning, each level is based on the level before it. Bateson operates in his learning model (Figure 11) with learning as transcendence of levels of reflection taking place on the basis of related layers of context (meta-communication). In Learning 0 + 1, there is a direct relationship between the learner/subject (S) and the object which has to be learned (O). At this stage of learning, there is no reflection taking place – a response is simply accepted. The first level learning occurs when alternatives are reflected upon in order to decide the correct choice. Learning at Level 2 is characterized by a reflection that leads to a corrective change in the set of alternatives from which the choice is made. At this level there is a systematic reflection on how to solve a problem, and the learner is conscious about the fact that he/she is learning. In this second-order learning, we learn about contexts sufficiently to challenge the standard meanings underlying our habitual responses. Thus, using second-order learning, we find ourselves capable of transferring our learning from one context to the other. At Learning Level 3 there is a relationship of reflections, in relation to reflection in learning. At this level the learner has a reflective attitude to how he/she him/her-self approaches learning. This level of learning usually happens outside concrete contexts. Level 4 is difficult to handle in reality (Sørensen, 1999; Raelin 2001).

Kolb (1984) postulated that learning occurs in a cycle in which learners engage in and then observe and reflect on experiences, assimilate reflections in a theory, and then deduce implications for future action from that theory. Kolb’s activity-reflection learning model is expressed in a learning cycle which starts with an initial experience and activity (Concrete Experience, CE). Based on reflection and observation (Observation and Reflection, OR) made on the initial experience and activity, a concept is formed (Abstraction and Conceptualization, AC) which can then lead to experimentation and new experience (Active Experimentation, AE). OR is most closely allied to 'negotiation of meaning' or 'initial understanding'.

In this connection, of particular interest to the work under consideration here is the mapping of reflection in professional practice in the sense of Schön (1983) to Kolb’s learning model. Schön’s concepts of reflection-in-action can be seen to be included within CE expressing the reflection which expresses our use of tacit knowledge. Reflection-on-action can occur in both the OR stage, where it may range from just the noticing of the significance of an experience, to naming the problems or questions that arise out of the experience, and in AC where usable concepts or hypotheses are generated. Reflection-for-action (when someone reflects to plan what they intend to do to confirm an understanding) occurs in the AE stage of Kolb's cycle where the implications of concepts are tested, and in AC stage in the formation of hypotheses (Brown and McCartney, 1998).

To avoid repetition and to render smooth flow, aspects of the organizational learning models developed by Argyris & Schön are dealt with as part of the presentation on the proposed learning approach below.

6.3 Learning and Process Improvement in Reflective Steps

In this section, we present a collaborative reflective learning model proposed (as part of the Reflective Steps approach) for software process improvement based primarily on the first two learning levels of Bateson’s comprehensive learning and the communication model discussed above. These levels of learning also correspond to aspects of the technical reflection (testing the validity of the methods and processes of use) and practical reflection (identifying and learning patterns from practice stories retold) types identified by Taylor (2004). To address the emancipatory reflection proposed by Taylor, one may consider introducing a third level that involves analysis of power relations, moral and ethics in the process of methods and process development. However, treatment of the third level may require is beyond the scope of the current work.

In practical terms, what is proposed is a multi-level reflection cycle (depicted below in Figure 12 and Figure 13) as a support infrastructure for continuous learning and communication, and the use of reflective workshops and reflective journals as a supplement to conventional methods of workshops and documentation. To render smooth flow in the presentation, the section starts with a brief discussion on collaborative reflective learning. This will be followed by presentations of the levels of reflections proposed together with the topics of reflection dealt with at each level. The section concludes by presenting the RS workshop proposed for practical use.

> In this connection, it is also relevant to note the following. Although the use and application of the model in the context of software process improvement is more emphasized in our discussions, we believe that the model developed is equally applicable to project plan improvement and software use improvement as well.

6.3.1 Collaborative Reflective Learning

As discussed earlier in Section 5.1.2 and 6.2, reflection is a key to professional preparation and development. Most of the examples cited in the literature in this connection, however, tend to largely relegate notions of reflective practice to the realm of individual learning. This predominantly relates to reflections that focus on the decisions that professionals make minute by minute in their practice (particularly, reflection-in-action). Such processes are also mostly limited to personal level. Even when a novice professional may be interacting with an expert mentor, the emphasis is on the reflection that each does and the influence of that reflection on each one’s individual practice. The type of arrangement that we are dealing with here, however, is more collaborative – supporting the social discourse among developers during software design.

Collaborative reflection as used inhere extends beyond the realm of individual learning in isolation. It occurs when two or more individuals, through a process of inquiry, work together to improve their own professional practices and programs in which they are jointly involved. In a typical software development project environment, the most effective way to encourage such reflection is to give a group of the project staff say a design artifact or a common process experience and ask them to work on same iteratively through a design-critique-reflection cycle. Collaborative reflection is more than simple discussion of a common idea. It is a prolonged joint work on the continual process of improving one's practice and the commitment to help others improve theirs (Osguthorpe, 1999). Criticism in the form of work product critiques is an important instrument commonly employed to realize group discussions and influence group decision making.

In the case of product design, for instance,

> social interaction during team work can influence individual perspectives and participative joint group decisions on the various aspects of the system. In this process, both critical reflection and negotiation are important instruments to clarify misunderstandings, to enrich design artifacts, to resolve conflicts, collectively innovate and agree upon design options and courses of actions. Individual perspectives on requirements, design issues and methods brought into the project sessions by participants are continuously discussed, collaboratively constructed and co-constructed. This group interaction process helps for the requirements, design and related methods-in-use to dynamically evolve during the process.
>
> The process starts with some sort of requirements, plan, design, etc. (the initial version). It then works from there by adjusting, altering, including new elements based on peer feedback and experience with the earlier version.

Collaborative reflection is most effective when participants:

- are invited to pose their own questions;
- differ in their professional roles and responsibilities;
- embrace the norms of reciprocity inherent in collaborative work;
- view collaborative work as one of their basic professional responsibilities; and
- take risks associated with their own practice, and extend the results of their reflections beyond the original group.

Among the benefits of collaborative reflection is that it is a kind of process not only for the individual aspect but also for the social aspect of learner-learner interaction. It provides a kind of learning process through which members in the community interact with each other. It is influenced by members' social participation and interaction. The activity to compare their own thinking with those of other learners would lead learners to be more articulate themselves so that learning does not naturally occur without reflective thinking.

The collaborative reflection approach proposed in Reflective Steps attempts to create an enabling platform that extends the conventional reflective practice at the individual level to a group level thereby supporting project and organizational levels of learning. The approach facilitates discussions and debates at first group and project levels and then at organization level. For this purpose, required repository and communication channels are created to capture and share the perceptions, debates and resolutions around project performance, design, and application of processes and methods.

In the case of design level collaborative reflection, for instance, the design representations are presented to stakeholders or collaborators for joint reflection. As expected, initial versions of the design are often incomplete. Through a means of critiquing, which reminds designers of other points of view (Avison et al., 2001), collaborators identify portions of the requirements that have not yet been understood and/or portions of the design that need refinements. In this manner, the reflections are made explicit through and after the critique session. In the process, shared understandings, contexts and the resulting design are accrued incrementally. Through such cyclic processes of design-critic-reflection, designers both evolve a representation of their design and gradually construct and accumulate criticism as articulated knowledge. This shared understanding helps the designers co-evolve individual understanding of a problem and a solution, and increase the knowledge about the design domain (and this learning).

6.3.2 Reflection Cycles and Levels of Learning

In summary, in Reflective Steps, we propose two levels of learning for process improvement (as depicted in Figure 12). Following the naming conventions in published literature, these learning levels are referred to as Single-Loop Learning (SLL) and Double-Loop Learning (DLL). While SLL is employed to improve practice incrementally by introducing corrective measures within the given processes and goals, DLL is employed to improve practice radically by questioning premises based on feedback and by improving given processes and goals.

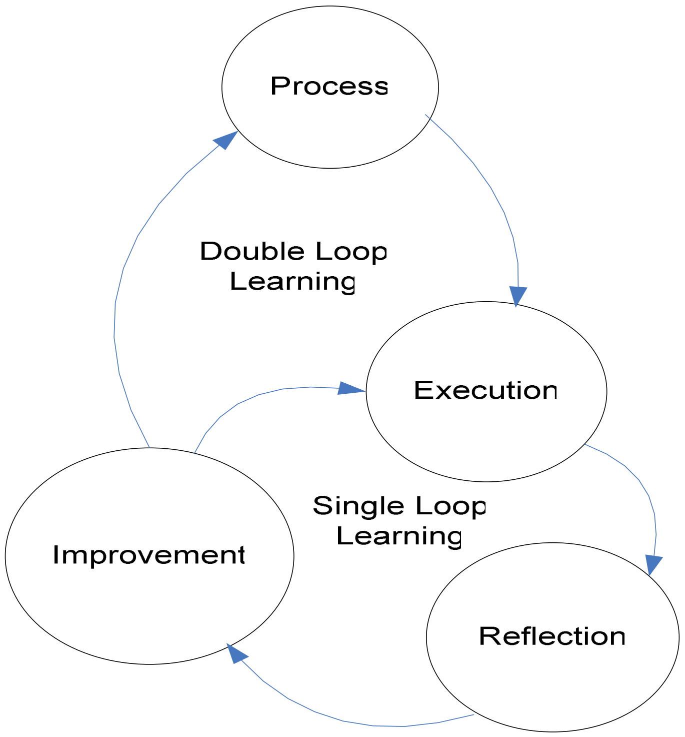


Figure 12: Reflective Steps: Multi-Level Collaborative Learning Cycle

The base model for the reflection cycles at each level is adopted from Kolb's learning cycle (Kolb, 1984). As discussed earlier in Section 6.2, in the Kolb's model learning occurs in a cycle of Concrete Experience (CE), Observation and Reflection (OR), Abstraction and Conceptualization (AC) and Active Experimentation (AE). To serve the purpose of collaborative reflective learning discussed above, we have modified Kolb's learning cycle that involves CE-OR-AC-AE into an iterative cycle of Action-Reflection-Improvement model. This modification also takes into account the mapping of Schön's concept of reflection-in-action to the Kolb's learning cycle discussed earlier. While the Action in the revised model may be seen to include aspects of CE and AE action, Reflection and Improvement of the revised model substitute OR and AC respectively.

The reflection process proposed may be typified as a continuous cycle of planning (reflection-for-action), execution (action that involves reflection-in-action) and feedback based assessment (reflection-on-action). For instance, with a software process as an object of reflection, the cycle involves,

- establishment of a process,
- application of the process for the increment,
- reflection on the process used for its effectiveness in achieving desired results (identification and selection of key activities), and
- application of improvements or adjustments of the process for the next increment.

The cycle is repeated iteratively. In the case of design, each cycle begins with a reflective comprehension of the situation that demands the action of the practitioner (requirements understanding). The actions taken produce design artifacts and testing of the artifacts (back-talk in the sense of Schön). The results of the actions may be different from the planned ones. Back-talk leads to reflection, which, in turn, becomes a predecessor of new actions.

In a collaborative reflection session, a typical reflection exercise on an issue is based on the interplay between the identification of an issue to be addressed (posing questions), critical discussion on the issue, and then proposing an action strategy to address the issues.

According to Bateson (Turner et al., 2006; Sørensen, 1999; Raelin 2001), the Level 0 learning, "not learning", in an organization becomes evident when individuals are isolated, fail to receive feedback on their actions and fail to receive and/or process new information. Within the existing local setting, individuals were observed to operate mostly at the level of "not learning", as identified by: the lack of feedback and project control mechanisms; the featuring of a 'them and us' relationship (disconnection) between project staff and management, between project staff and users, between software developer and client organizations. It is only in some of the cases that we observed feedback mechanisms used to monitor project progress.

The SLL is more to take a corrective action to ensure adherence to procedures, in the sense that the information obtained as feedback is used to correct errors in an attempt to bring about the expected performance levels. For instance, during project review, by comparing intended and actual performance, the variance is analyzed. The results of the analysis (often based on determining cause and effect) are used to take corrective action to address the variance. The reflection at this stage may involve auditing the existing competencies and skills to assess their adequacy to perform project tasks within the given constraints. As a result, such corrective measures as providing training on how to use tools, on how to apply guidelines, or exchanging roles, or putting additional resource, etc. may be considered.

Where such corrective actions make no significant difference (do not help improve the situation any more) in terms of reducing the variance between the intended and actual performance, questioning and changing the procedures and processes may be considered. In a changing context of software development, the plan or targets set by the plan as well as the conditions within which they were set cannot be assumed to be not changing. Under the circumstances, therefore, either a forward looking anticipation strategy or a DLL system must be employed to succeed.

DLL offers a higher level learning opportunity that allows for the adjustment of the input variables to the process as well as the adjustment of plans that are used to dictate the performance standards. It also incorporates the SLL. It is initiated when corrective actions taken at the SLL level do not really help in realizing intentions or when intentions need revisiting because of changing circumstances. The ability to respond to change and alter performance standards, process strategies, redesign of products, etc. encourages adaptability and improves the chance of sustainability. The DLL system enables the project to become more adaptable and to do so more rapidly. This adaptation means that the project is capable of learning and continuous improvement in a search for better performance and goal achievement. DLL also provides an opportunity for long-term learning. In contrast, the SLL system only focuses on short-term adjustments during the duration of the project that are likely to increase the chances of meeting the objectives of the current project.

In their work with system developers, Mathiassen & Purao (2002) emphasized that double-loop reflection, which questions assumptions and values aligned to the project, creates knowledge that is ‘highly local, specific to the context’. That is, instead of simply trying to build capacity that would upgrade the skills of practitioners in applying the techniques and methods suggested for systems work, one may need to go beyond. One needs to question whether or not the assumptions and theoretical underpinnings that underlie these techniques and methods are suitable to or take into account the specific project settings. This was, in fact, partly the motivation for this study.

Furthermore, lessons learned from the DLL system must be accumulated and used to improve or even challenge the process at company level and related premises. DLL is in a way a reflection that encourages standing back and questioning the presuppositions attending to the problem. By so doing, it enables to change the theory-in-use to improve performance as a result of an enquiry into the situation and questioning the norms and values by which the action is judged. DLL goes beyond the cause and effect determinism of SLL that aims at getting better at executing the plan, to questioning the premises of the plan and the values used to judge action outcomes. As such, it involves reflection on the method selection, the adaptation process itself and on redesigning of the product.

The idea of double-loop learning lies at the heart of much of recent thinking on both individual and organisational learning. Such DLL is typified by the learning cycle proposed by Argyris and Schön (1996). It raises the challenge of nurturing individuals who can deal with uncertain and changing environments; develop abilities to question, challenge, and change assumptions and behaviours.

6.3.3 Reflection Topics at Various Levels

In Reflective Steps, collaborative reflections are held at various levels and on various topics. The diagram in Figure 13 below tries to synthesize these various levels with emphasis on the topics of reflections.

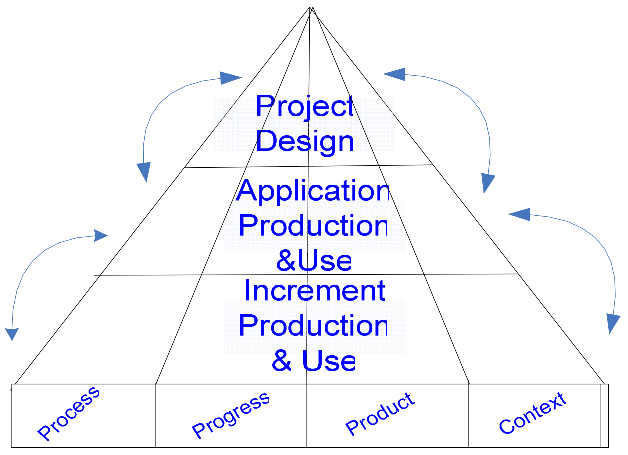


Figure 13: Multi-Level Collaborative Reflection Topics

As shown, the model tries to provide for the reflections depicted in the integrated RS model discussed earlier. In particular, it involves reflections within the design and development of an increment, reflections at the end of an increment development, reflections at the end of an application development and reflections at the end of the project. The reflection topics at various levels may address reflection on product (the software delivered), on process (method of work), on project progress (status of activities), and on context (business environment, technology options, contract administration, etc.). First, reflection on the product is about the extent to which the product meets user expectation in actual use. This type of reflection tries to address issues such as the following. What did the users and sponsors think about the product? What should be improved, added, and removed? What are the priority directions for product evolution? etc. The reflection on progress relates to the reflection on project progress issues such as the following. How does actual performance compare to the plan? Are we ahead or behind schedule, and why? What are the corrective measures that need to be taken to enhance performance? This may lead to revision of plans in terms of scope change, mobilization of more resources, etc. The reflection on process examines the activities, sequence of activities and techniques used for software development.

Looked at from a different angle, the topic of reflection may also vary with the main processes of Reflective Steps identified in Figure 4. For instance, at Project Design level, the reflection is characterized by organizational change matters and business matters. It is held among project sponsors, business experts, software companies and external consultants. At the Application Production level, the subject of reflection is more technical in nature. It has mostly to do with requirement understanding, sort out design issues and options, programming options and use of methods. At the Application Use level, the reflection subjects relate to embedding of the software developed in the organization, training of users on the operation and utilization of the newly developed system, and generating of new requirements for the next round.

As such, reflections at the various levels are done by different, but overlapping, categories of users. For instance, at Application Production level, while the reflection is done mainly among development team members, other stakeholders particularly users may join them in matters related to product quality. Likewise, at Project Design level, representatives of the Application and Increment Production and Use levels do take part in the reflection on matters relating to project priority setting and scoping, contract negotiation, etc. In this connection, it is relevant to note the fact that often communication related problems feature when mixing these levels, and as such this needs to be carefully handled during such overlaps.

At application level, the project teams may pause to check if the course is right for the project, and to reflect on their experiences in order to conduct short-term improvement actions. The reflections and related learning at this level are as such limited to SLL. On the other hand, reflections at project level (including those that are made at the end of an application development and use) are more critical and mainly serve the purpose of DLL.

By suggesting such multi-level and integrated approach where learning is cascaded bottom-up with well defined links, we offer a perspective on software process improvement and software development practice improvement that exploits project experience in a complementary manner to other popular top-down approaches.

In Reflective Steps, each product release (be it an increment or an application) is a mini-project. As such lessons learned from each mini-project is shared up the ladder. To operationalize this, as shown in the model proposed, the outcomes of the reflection on the various topics (product, progress, process and context) are gathered, interpreted, consolidated and rolled from bottom up (from increment to application to project levels). Such consolidation reveals key issues from the reflections at the lower level for the purpose of creating a shared understanding, guidance and facilitation of decision making at the higher level. Likewise, the outcome of the reflection at higher level (which is done based on the feedback obtained from the lower level) is shared down the ladder in the form of guidance, prioritization, scoping, resource reallocation, etc.

Learning Repository

It is important to note that learning does not simply occur all at once, it is built up from the step-by-step reflections made on the actions and reinforcement being taken throughout the process. Better understanding of the requirements, the feasibility of design, the suitability of the software increments, the need for process improvement, the need to re-prioritize applications and/or redefine project scope, etc. that result from the reflections represent the lessons learnt (the learning products). What has been learned from the reflection needs to be implemented as an improvement or change either in the product, process, progress or premises. Otherwise, it is difficult to say that learning has occurred. Both the learning product (lessons learned) and the learning process (process of reflection, reinforcement and the implementation of the outcome) need to be stored to facilitate use and application in the current project (by project staff). The storage and maintenance of the learning product and the learning process also help others that will be involved with the system maintenance and upgrade at a later stage. As required, the process-related lessons captured will also serve in the effort to improve processes at organizational level for use in subsequent projects. That is, the stored materials will serve as input in the process of establishing or revising the methods-in-use for the organization.

To support the foregoing, storage of learning products (lessons learned) in terms of 'memories of individuals' or in some flip charts and maps may not be enough. For this reason, Reflective Steps suggests the creation of a common repository and communication channel to capture and share the perceptions, debates and resolutions around project performance, design and use processes and methods. In particular, the use of a reflective project journal[15] as an experience repository tool is suggested. Such a reflective journal is to be regularly updated by the project management using journal summaries written at the end of each collaborative reflection session. The importance and relevance of such experience repositories were already established by such popular approaches as the Factory Experience approach (Basili and Caldiera, 1994).

With such repositories and consistent practices of reflective activities entailed at the various levels, the project team and the management can develop an integrated view of project progress, the product developed and its use, as well as the processes and methods employed in the development and use of the software. What is more, such a repository can be used in the establishment (in case there is none) and/or improvement (in case there is one) of processes for consideration in subsequent projects. Such a repository also facilitates the sharing of experience across projects that are active at any one time within the organization. The sharing may also be extended between organizations (as the case may be) to provide cross-project or cross-organization feedback to further facilitate learning between projects and organizations.

[15] As indicated, the maintenance of reflection journals enables the acquisition, storage and retrieval of project experiences and memories, thereby providing directions to the development and improvement of the process. Journals come in many types ranging from log books (recording tasks and performances) and personal diaries (recording thoughts, intentions, desires and activities) to learning journals. This study concentrates on learning journals that are also used as tools to develop reflection skills.

6.3.4 Reflective Steps Workshops

Among the main problems frequently encountered in the course of developing soft skills by both practitioners and students were the lack of skills in conducting system development workshops and managing meetings. Over the years, attempts were made to use customized versions of JAD techniques for conducting software development workshops. However, difficulties were reported by students and practitioners involved in the process. Among the complaints reported were the following: the procedures are expensive; at times they are relatively heavy on processes; they are more helpful for requirements elicitation and not for iterative design activities. For this reason, in order to provide operational support to some of the features of Reflective Steps, the development of workshop techniques and meeting protocols for use with Reflective Steps was initiated. As usual, the proposal was still an evolving one based on real-life experience in both software development and management practices. What follows is a brief description of the status of this work at the time of this writing.

As discussed earlier, collaborative reflection is realized in the form of facilitated dialogue among team members in a workshop setting. It is a form of face-to-face session to openly talk about and critically assess the way of working by coming together and where this helps participants achieve their objectives. Where there are things that were done better, the sessions are held to discuss and share the factors that have contributed to such strength and how to maintain them in the subsequent course of action. Where there were weaknesses, the sessions are held to investigate the root causes of the weaknesses and discuss ways of resolving them. The focus in general is not to discuss problems but rather to explore ways of improvement. In the remainder of this subsection, the various aspects of the proposed workshop are presented.

(i) Facilitation

To render a purposeful and meaningful workshop and manage transactions for the purpose of facilitating effective communication in the sense of Tan (1994), for each workshop, people with such specific roles as a facilitator, a scribe and a process expert are to be assigned as organizers of the reflection workshops. The project leader is the most eligible candidate for facilitation. The scribe is a software engineer assigned by the project staff or project leader to record the proceeding of the meeting/workshop focusing on the substantive issues covered by the session. This may include: issues discussed, decisions made and action items identified. It is relevant to note in this connection that a scribe is not there to play a secretarial/clerical role but that of an active participant with this specific role. The role of the scribe may also be shared or circulated/rotated among the team members (through rotation).

(ii) Process Steps

A three step process is suggested for conducting collaborative reflection workshop: initiation, conducting and wrapping up.

First, as part of the initiation a list of reflection issues on the topic of reflection together with necessary support material are prepared and circulated. Such a list is better circulated at least one day earlier to give participants time to think about and reflect on them or prepare their comments individually. Members should be encouraged to carefully review the documents circulated before coming to the meeting. In fact, it is preferable if they come to the meeting with their comments recorded on a simple free-format feedback sheet. Such forms may be used to record comments on unclear points, contradictory points, serious cases, point of views, experiences to be shared (like retelling stories if any, for instance), etc. on each of the reflection issues. As required, specific formats preferred may also be designed for this purpose, as an alternative to the free format feedback sheet. Where this is the case, it is recommended that the form be circulated together with the agenda.

Next, as part of the conducting step, the meeting is held to discuss and collaboratively reflect on the issues. The feedback sheets may serve as support materials in the discussion sessions. The discussions of the meeting have to be recorded. In addition, it is also useful to collect the feedback sheets from the individual members. For more on meeting and deliberation techniques, see the discussion under '(iii) Conducting Session' beyond. The use of ‘feedback sheet’ is an effective means particularly in view of the limited time to be allocated to the workshop. It also gives a chance to all participants to communicate their points of view. It also allows the quieter, more technical people to develop their comments on the issues. In most meetings, such people are often interrupted and dominated by the more outgoing, vocal types. On the other hand, earlier reminders also give participants time to create more thoughtful responses on the issues circulated.

During the course of the discussions and deliberations, each member can update the feedback sheet entries by including additional information learned in the meeting (in the form of clarifications, questions, etc.) against each of the issues. The free format also enables participants to record complex issues or problems related to the issues of reflection (analysis of the problems, description of solutions, etc.) in text and/or figurative forms. In this way, the free format serves as a draft reflection journal at individual level. Similar techniques have been effectively used in the course experiments conducted as part of this research, particularly in the case of the business process redesign work reported in Chapter Four and the requirements definition in the case reported in Chapter Seven.

Finally, in the wrap-up step, based on the feedback sheets and the discussion record as input, the reflection outcomes are summarized by a team composed of the facilitator, the process expert and the scribe in the form of "lessons learned". This is a sort of brief reflective journal writing to systematically flesh out and document the discussion outcomes by key issues and topics. While we do not insist on a specific format to be followed, journal entries under the following headings are suggested.

- Strengths: things that were performed well and factors that have contributed to that and ways of maintaining them.
- Weakness: what went wrong or did not go well and the causes for same.
- New things to be focused upon.
- Major problems to be addressed, things to be improved and ways of addressing them (reinforcements or corrective measures to be taken).

- Action plan for the measures (including assignment of responsibilities).

It is important that the entries under these headlines are brief and to the point (preferably, the summary report from one meeting better be accommodated on a page). Otherwise, the chance of it being read is low as most members may not afford the extra time required to read a lengthy document. However, any form of presentation material and documentation used during the collaborative reflection (charts, drafts, the feedback sheets with the comments from the discussions, etc.) can be annotated and attached to this one-page summary journal, for reference as required. All these, together with the summary reflection journals to be prepared at project level based on periodic reports and project postmortems should be filed in the project experience repository (project journal).

The summary report then needs to be circulated to all members and other interested parties before the next reflection meeting/workshop for both sharing of information and checking the correctness of the summary. From our experience, creating a means of communication between meetings is particularly challenging. So is the means of ensuring the timely use and application of lessons learned from the reflections. From what we have seen, creation of an electronic distribution list and a project server together with extensive use of email might help for the communication.

(iii) Conducting Sessions

Usually it helps to open each meeting with a period of reconnecting and chatting, often on topics not related to the project, mostly social issues. This is very important in lubricating or easing the tension usually imposed by formal meeting procedures. Some of the conversations at this level might also relate to sharing experiences and information which directly contribute to the issues to be discussed in the formal agenda. From experience, allotting about 10 minutes of time here may be enough (see beyond for more comments on managing meeting time).

The reconnection and chatting during the first few minutes should be followed by an introductory session on the agenda items (which are already sent out to members ahead of time). The first agenda item should relate to briefing on the developments that took place since the previous meeting. This is an open session where members recall the discussions they had in the previous workshops and brief each other on the developments since. The briefing and information exchange at this stage should be deliberately kept informal and unstructured. The whole purpose is to reconnect to the issues addressed in the last meeting and recall some of the noteworthy issues or important lessons learnt or decisions made, and to report (give feedback) on the effect of the lessons learned on the actual project operation. The briefing session should also be used to share any valuable information (for instance, telling stories where someone came across or noticed some good or bad experience or new information) worth sharing with others. For this purpose the reflective journal from previous meetings, already distributed to members by this time, may be used as a reference. Such sessions also provide a context to follow up lessons learned and difficulties experienced in implementing resolutions of previous meetings.

In every meeting, once the briefings and information updates are dealt with, the critical reflection sessions on the main agenda items follow. The topics that make up the main agenda items are usually issues selected by the facilitators (in close consultation with participants) for reflection from any or combination of the reflection topics related to the project and production processes: progress, process, product and context. Such issues may also crop up during any of the reflection sessions. It is recommended that the scribe maintain an issue log to keep track of and line up discussion issues.

The reflection session on the main agenda items is a sort of purposeful dialogue that is designed to extend the understanding of the domain knowledge, software development process and method concepts, or design issues within the group. Individuals may share their views based on the information recorded on the feedback forms. In some of the cases reported elsewhere in this report (Chapter Seven), a simple and free-format 'presentation-and-critique' dialogue session followed by a prompt and stimulated reflection session were used. In this case, individuals and subgroups each take turns to share their views in the form of presentation or critique. When one group presents, the others may ask clarifying questions or forward critiques, etc. The groups may then switch roles, reversing the presentation and questioning sessions. This approach worked well both in the case studies and course offerings at CTIT and AAU.

> According to experience, to render a purposeful and focused reflection, use of prompts/questions to stimulate conversations into a collaborative reflection is recommended. In particular, probing the meeting to questions such as the ones described in Section 7.1 may help.

Before closing the workshop session, the last item on the agenda is discussed. This usually is a recap session – it relates to summarizing the outcomes of reflection sessions (key issues), and to articulating and agreeing on the corrective actions that have emerged from the discussions, as well as assignments of specific responsibilities.

> On the whole, in connection to allotting time to each agenda item, it is important to emphasise the concept of time boxing. The overall meeting is better kept within a two-hour limit to have effective sessions, and from experience the best time of the day is between 4:00pm and 6:00pm. Having said this, however, there may be exceptional cases that may require more meeting time. In such cases, instead of extending the collaborative discussion sessions for the whole group, better still is to task a special working group to deliberate in detail on the issues in separate sessions and bring back the outcomes for sharing and reflection at the whole group level.

> In addition to time boxing, an issue that also needs to be regularly checked is the proportion of time spent on irrelevant factors and important factors. On the practical side, it is also important to note that giving and receiving feedback can be time-consuming, so one needs to be realistic about what can be achieved.

> Among the major challenges in conducting the meetings is also the need to balance on regular bases, the informal communication that contributes to the shared experiences that are the very foundation of the project community, with that of the formal and purposeful dialogue and communication that extends the understanding of the concepts around which the meetings are organized.

The role of facilitators to address these (aforementioned) aspects is vital. The use of feedback forms stated above is introduced to partly address these issues.

(iv) Writing Reflective Journals

As repeatedly indicated, reflection is an important tool for learning from experience. We reflect in order to learn something, or we learn as a result of reflecting. According to Moon (2000), we learn not only from the 'in the head' reflection but from the process of representing the reflection itself in some form. As the saying goes, "you don't know what you know till you have written it down". Here comes the importance of reflective writing for reflection. To this end, in Reflective Steps the use of reflection journals is considered for fostering and supporting communication and learning in a collaborative development environment.

> Although other similar studies Jepsen et al. (1998) have used diaries to support the system development management, the work here extended this approach in a number of ways. First, instead of diaries, the use of reflective journals is considered. Second, the process of writing the reflective journals itself is an activity in the Reflective Steps process that promotes structured reflection at individual, group, project and organizational levels. Third, the application area has also been extended to include software process improvement in addition to software development project management. The approach suggested inhere also tries to make use of such common group communication infrastructure (meeting) for this purpose.
>
> Another related mechanism is the use of logs. On a project log you might write down the times and days when you performed a project activity. A log is simply a record of events. The journal as suggested inhere is designed to help members organize their reflections on the project and the production process, to document members' work and experience for self-evaluation during and at the end of the project, to provide a place for members to write questions and comments for the project team to discuss.

(v) Periodic Review

In addition to the weekly reflection sessions, periodic (for instance, monthly at project management level and quarterly at steering committee level) attempts must be made to further digest the findings of the reflective exercises. In this connection, it is also relevant to note that depending on the size and complexity of the project, where there are large size subteams, additional facilitation teams may be assigned at the subteam levels. Where this is the case, facilitators, scribes and process experts from each group should come together every month for the project level reflection. Likewise, the project level facilitator, the scribe and the process expert have to participate in the quarterly meeting at the steering committee level.

At the end of the project, the reflective journals are collected into a project journal which will serve as an experience repository and for common reference. For this purpose, as part of the post-mortem, a reflective project journal should be prepared consisting of careful descriptions and evaluations of what happened and what should or could have happened. Such a journal serves as a source of input for the project portfolio management at Project Design level. An additional utility may also be developed to facilitate an efficient storage and retrieval system, particularly to enable efficient tracing of individual design issues and decisions.

CHAPTER SEVEN

7. Experiences on On-going Projects with Reflective Steps

In this chapter, we report on additional and ongoing research activities (field work) with Reflective Steps and experiences thereof. In particular, we report on:

- experience in teaching software development using Reflective Steps insights; and
- experience from the ongoing project at Organization B (the third phase project that aims at the procurement of an off-the-shelf complete insurance application software and its customization per the requirements of Organization B).

7.1 Recent Experiences in Teaching System Development

7.1.1 Background and Motivation

One way to address the competence building for software development professionals is through work-based training (through reflective learning at various levels in the project work) as pointed out in the preceding chapters. Another way is through academic learning – through the various courses offered by higher learning institutions as part of the academic programs that prepare graduates to join the software engineering profession. According to the findings from the survey and the researcher's years of experience in teaching system and software development courses in local institutions of higher learning, the existing programs are strong in technical aspects and weak in social aspects. As documented in Chapter Three, both instructors and students lack real-life project experience. Thus, an attempt was made as part of the current research to explore to what extent Reflective Steps concepts can help to improve this situation.

As elsewhere, in earlier times we only used textbook examples and cases (in books mostly published in the West) to integrate practicum (course project or case work attached to the lectures) in teaching programming and systems. Such cases were particularly used to apply the classroom concepts. Applying such techniques, we noticed from early on that we only taught our students the steps of the 'dances' as published in the text books without contextualizing them (to both the audience and the dancing stages). With such approaches, not only were we unable to teach them how to dance, but we never created an opportunity for the students to try out the steps (experience dancing in the real sense). For that matter, most tutors themselves never had the chance to dance in a real-life setting. Under the circumstances, therefore, it was unfair to expect the students to perform the dance as they join the professional practice (the real dancing stage). This is how the author tries to explain the earlier complaints (obtained during the survey) from the companies on the practical skill deficiencies of graduates.

With more exposure to local cases, though not in an institutionalized form (in a manner that guarantees sustainability), attempts were made to first cite local cases and then to use them in the practicum. The motivation to explore the integration of real-life and active projects in course practicum, as reported in here, partly came from the experience at the School of Information Science and Technology (SIST) at AAU (reported in Chapter Three). At SIST such arrangements were tried out successfully (at least initially) in the form of incorporating industry projects as a course in the academic program. However, the motivation and interest to take this approach as a research partly came from the inspiration by the works of Greenbaum and Mathiassen (1991), Drohan et al. (2006) and Hadin et al. (2007).

> "In our profession, we seem to saddle our students with so much talk of theory and especially methods, that they become confused when they actually have to apply them. While we compensate for the students' lack of systems experience, with the old "stand-by", the case study approach, we often fall short of being able to actually integrate theory, method and experience". (Quoted by Greenbaum and Mathiassen (1991: 524), from a letter to the authors by a colleague).

> "The process of teaching is, after all, a lot like the process of systems development. We never really know what the end result is going to be like and how it is going to be used! We can certainly not expect students to become competent systems developers through a series of step-by-step instructions, any more than we can do reasonable systems development in this way. Teaching, as we know intuitively, is helping students make their own decisions. And it is this

process—the process of exploring and testing—that can give students a focus on experience and the context of experience that they are missing." Greenbaum and Mathiassen (1991: 526)

"If students are encouraged to set their own problems, and be aware of their experiences as they do so, then they are, hopefully, taking steps toward managing both the learning process and the systems development process, as well." Greenbaum and Mathiassen (1991: 524)

Our work also draws on a metaphor of 'steps and dancing', (adapted from Turner, et. al. (2006) and customized to the specific situation under reference in here). In Turner et al., a metaphor of 'steps' and 'dance' was used,

"to critique individual learning experiences in organizations, to explore the role people play in inhibiting learning in organization and to explore theories of individual learning as "theories in use". The "steps" imply a fixed form which constrains the individual within the confines of the job role, while the "dance" relates to fluidity and flexibility which enables individuals to express movement and therefore learning." (Turner, 2006: 398).

In the research being undertaken by the author,

the 'steps' and 'dance' metaphor is used to critique the training of software development courses in the classroom environment disconnected from reality and to promote teaching methods that combine classroom lecture and lab exercise with a practicum in real-life project context. The 'steps' imply the teaching of software development techniques as a series of fixed and prescriptive steps involving requirements, functional specifications, code, testing, mostly supported by examples and exercises provided in textbook and simulated classroom projects. The 'dance' on the other hand relates to the actual practice (performance) in real-life situation to learn more about the steps and application by adjusting the steps flexibly and inventing possible actions based on feedback and intensive interaction with the events and contexts.

7.1.2 The Project in Brief

As indicated in the introduction above, this is a brief report on action research in the form of field work conducted to test the applicability of Reflective Steps in the teaching of system development courses. The work consisted of three graduate level teaching experiences supported by real-life project-based practicum, where the author actively participated as an instructor and guide in software engineering, system development and software project management courses. These courses were conducted over a period of three consecutive academic semesters and with different batches in different departments: Software Engineering at the Department of IT (CTIT) during the second (spring) semester of the 2005/6 academic year, Software Project Management at the Department of Computer Science (AAU) during the first (fall) semester of the 2006/2007 academic year, and Information System Development at the Department of Information Science (AAU) during the second (spring) semester of the 2006/2007 academic year. Participants in the study were post-graduate students. The work-based teaching/learning method involved extensive reflection sessions, where the researcher played the role of a coach.

Although there were developments from the first teaching semester to the second and then to the third, to economize on space, the cumulative experience is summarized in the remainder of this section. The detailed story, including the experiences of both students and instructors that participated in the work, is being separately compiled for experience sharing and reporting purposes.

(i) Project Design

The skill development followed the basic tenets of Reflective Steps: taking incremental steps to the destination and through reflection at each step adjusting the strategies adapted to get to the destination. In this case, the destination relates to building software development skills required in real-life project environments. In clarifying concept and building skills on adopting and customization of methods, we went step-by-step. In between the steps, the class as a whole, instructor(s) included, reflected on the learning experience. Throughout (from adjusting course outline development to the assessment), active participation of students was encouraged.

Over and above short-lived assignments aimed at teaching specific skills, the course design included course-long projects for the practicum with real-life client and work environments.

Contrary to earlier experience, where the course-long group project cases were designed by the instructor(s), in this case students were given the opportunity to actively involve in the process of designing the course project and formation of project groups. Each group was also guided to make the necessary working arrangements with the client that hosted the real-life project and amongst team members. Admittedly, this was not initially welcome by some students and clients (because of the seemingly conflicting perspectives and concerns discussed below), but gradually through reflection and review, these were resolved. In this connection, as applicable arrangements were made for previous year (older batch) students to share their project experiences in special sessions arranged for this purpose as part of the regular classroom lectures (they were invited as guest lecturers). This helped a lot in terms of story sharing and encouragements for the new batch particularly in terms of comprehending what could be achieved in the course project.

In the first lecture session, we discussed and agreed with the students on the approach to be followed in the actual conduct of the course. This was done at two stages. In the first stage, the course outline developed for the semester by the instructor(s), based on the course descriptions and objectives as defined in the curriculum, together with the proposed teaching method, were presented to the students for comments and discussions.

After this was done in the first session, students were asked to study and discuss the proposal amongst themselves until the next session where the proposal would be enriched and considered as a starter course conducting process for the semester. In the meantime, students were also asked to come up with ideas on candidate project cases from real-life projects for consideration in the actual conduct of the course. Tutors also brought real-life project cases selected with prior arrangement with respective clients and this was done mostly through professional contacts (not through institutional arrangements).

After making consultations with both students and client representatives, joint discussions were held in the second sessions (a third session was also held where necessary) to finalize the selection of projects and the assignments of groups.

Students were advised to use methods and processes published in textbooks as a guideline to get the process started[16], and then to reflect in action[17] to continuously adapt the software processes and methods for the different situations in the software projects.

As part of the group semester project, in the course of developing the work product required, students were asked to:

- investigate the methodologies and techniques available to them and the appropriateness of these methods to the particular situation they were going to work on;
- reflect on their experiences on a weekly basis – articulate the experiences that they went through in the class and in the project;
- share their reflections among the group members – to enable/allow them examine the way others in the team perceive the same experiences;
- reflect on issues related to the use of methods, non-canonical practices, interactions with users/tutors; etc.

Students were asked to prepare and submit, together with technical reports and the various system development work products, a reflection journal written based on the reflections made at various stages of the work. The reflection journal was allocated a weight of 10% in the final grading.

[16] The RUP as a starter method and Object Oriented Programming in C++ or Java, depending of the choices made by the students, were used in the courses.
[17] When students collectively engage in exploratory reflection and critique, they are in effect reflecting in action.

(ii) Project execution

Throughout, with regard to course projects, the instructor's role was limited to guiding the students to find their way out of the problems by their own. Classroom lectures were used to introduce available theories, methods and tools. Actual skill development took place in the projects where the classroom concepts were applied and tried out.

At the start of the semester, as the course outline was discussed, reflection was introduced as one of the teaching and learning methods. At this point, the meaning of reflection, the stages of the learning model were explained to the students and discussions were initiated to help students understand the concepts better. Even at this early stage, an about six-minute timeout session was allotted from each session to the students to share their understanding of the concept by talking to the person sitting next to them or in ad hoc groups created on the spot. This session was concluded with a joint reflection with the instructor on the concept and techniques suggested for reflection.

At the beginning of each class, once in the middle and at the end of the class, students were given timeout sessions to reflect on the concepts discussed and jot down notes that would serve as inputs to their reflective writing at a later stage. (The total class time was 90 minutes, hence the reflection time accounted for about 20% of the class time.) Students were advised to keep these notes and complete their reflective journals later in the day when they have more time. What they have jotted down while in class could be considered as a recording of their initial reflection-in-action, while what they write afterwards based on this initial version could be considered as a revised version of their reflection-in-action and an initial version of their reflection-on-action. In between classes, students were also encouraged to use this revised version and the course outline as a resource to reflect-for-action, by way of preparing themselves for the next class. This was continued for the first couple of weeks until students demonstrated a reasonable skill in reflection. In this connection, it is relevant to note that the students learned about reflection more by practicing it in their group reflection exercises. After that, we only scheduled timeout sessions for brief reflection within a classroom lecture session on as required basis.

Specifically, students were required to reflect both on the content (what they have learnt) and on the process of learning (how they have learned). As a result of the reflection on content, students were required to summarize what they have learned and compare it with the expectations they had jotted earlier. In addition, they were also asked to write a report for themselves on what they had really learned – any change in the understanding of concepts, techniques, and tools, etc. With regard to the process, they were asked to document whether the arrangement for teaching and learning was helping them to achieve the course objectives.

Reflections made in the classroom (in between lectures) were guided with prompts like the ones listed below. In fact, at the beginning of the course, students were given a working draft list of reflection prompts to adapt and use them in their respective settings. Students were particularly encouraged to update the list based on their experiences. In a related undertaking, work is already underway to develop such list of reflection prompts into a standard guideline to be used across courses.

- How did the task progress?
- What new skills/qualities/abilities did the students develop?
- What worked really well? What was the successful achievement this week?
- What went wrong and why?
- What were the major problems encountered? How did students try to address them?
- What needs improvement for better achievement?

At certain intervals, particularly at milestones in their coursework, students were required to make collective reflection at two levels: at their own team level and at the class level. The collective reflection at these levels could more or less be considered as ‘reflection on reflection’ as in the double-loop learning, but limited to the process of teaching and learning. For the collaborative reflection at the class level, students exchanged their team level reflection journals with the rest of the team. A thirty-minute reflection session was assigned to jointly reflect on the work of a student group at this level. At the end of the joint reflection session, each team/group was also encouraged and required to publish its reflection journal on the course server which was accessible to members only (for the first two courses, students created their own shareable resource using preferred groupware tools, while in the third course, the Moodle learning platform was used as a common platform for all). A copy of such a journal was also submitted together with the deliverables for review and assessment by tutors (mostly the instructors, but in some of the cases by practitioners from the client organization who participated in the project). Free formats were used for writing the reflection journals.

In addition to the monthly presentations made by each group in the classes, at the end of the semester students were required to demonstrate the work products developed in the course of the project. This was followed by full presentation of the process and the outcome in front of the class and invited guests. In the discussions that followed, students defended their work by addressing questions from the audience.

The actual conduct of the course was concluded by an assessment workshop between the tutors and the students. This was basically a forum for the students to evaluate the course and the tutors. The outcome of such discussions was to be considered in the assessment of the students' work and in the planning and offering of the course for the next batch.

(iii) Discussions

Analysis of the initial results of student work in all three courses showed that reflection has helped students to learn more, know more and appreciate more the use and application of various techniques in software development and project management. Among the key factors, according to students and staff involved in the exercises, in improving the quality of student reflection was the "time to reflect", reinforcement of the reflection through collective reflection and reflective writing as well as mentoring.

Over time, signs of gradually building a culture of reflection among the student groups have also been observed. Throughout, particularly as compared to previous classes and cohorts, we were able to observe noticeable improvements in the students' understanding of concepts, methods and techniques of system development. This was in particular glaringly evident in such soft skills as team work and project management. These were confirmed by the comments (both oral and written) from students (as indicated earlier, these details are being compiled separately).

Closer analysis of the situation revealed that most of the challenging issues revolved around addressing conflicting viewpoints of stakeholders and actors around the project. In particular, issues related to mixing and striking a balance between the following concerns were challenging.

- Student concerns: obtaining a good grade for the work.
- Client concerns: the project must be undertaken professionally.
- Instructor concerns: the use of appropriate methods taught in the course, balancing the technical and social aspects, contribute to the achievement of the project objective (both at the course level and at the product level).

Students also complained (rather expressed concern) about the time taken to write reflective journals on weekly bases. After the first month this had to change to monthly basis.

Due to the size of the class and frequency of reviews, tutors mostly gave feedback orally (without a supporting written document). Students also expressed the concern that the notes taken during the oral feedback session did not capture the points as a whole and in some of the cases they overlooked or missed vital points and essential aspects. Some reported that they had run the risk of being unable to retrieve or recall certain points. On the other hand, students very much valued and benefited from the feedback they got from their peers and from senior practitioners in the client environment.

Students also mostly complained about the problem of management and coordination, as well as the lack of active participation by some team members, and the difficulty to resolve conflicts whenever such problems arose within the team. Most students preferred to withdraw from the group or split or join another group. With more discussion and reflection on group work and increased awareness on the various stages of group development (forming, storming, norming, performing, and adjourning), these problems were partly resolved in the process in most of the cases. In some of the cases we had to reorganize the groups to resolve the problem.

Surprisingly, students had no problem with talking in meetings as well as in the classes, but lacked skills in conducting and facilitating workshops and meetings. What is more, students were also observed to use email and mobile phones for more communication in addition to face-to-face meetings.

(iv) Concluding Remarks

While the use of textbook examples and cases is still helpful in terms of building specific technical skills of students, these are not sufficient in terms of preparing the graduates for work in the local settings. It is realized that creating a situation for students to participate in real-life projects provides an opportunity where they apply acquired skills and knowledge towards the satisfactory resolution of the particular problem situation. It provides an opportunity where they develop an enhanced understanding of the particular skills and knowledge set, which soon becomes a typical solution that is applied in similar situations of concern in their working lives.

To facilitate increased learning from experience and to develop increased knowledge in problem solving, a reflective approach to learning is useful. The contention is through a step by step action-reflection-improvement approach premised on real-life project practicum, valuable lessons can be obtained on system development methods and approaches. In particular, the Reflective Steps approach demonstrated its potential in terms of supporting teaching methods that aim at developing skills required to address real-life system development problems. These include skills related to interpersonal communication, conflict identification and resolution, teamwork, and exposure to current technological tools and techniques. Both the group discussions and reflective journals prepared on the basis of the discussions served as better channels and medium of communication among the group members. The iterative group level reflections on both the content learned and the learning process had positive effects. The complaint on time notwithstanding, the advantage of reflective journals in providing an enduring record and reference point was widely acclaimed by students. That it can be viewed and reviewed for both project work and exams was identified as a great advantage.

Our experience as part of this study also demonstrated the ability and willingness of students to iteratively improve their perception and project work practices around these and similar issues.

However, for better results, both staff and students need to develop better awareness of the step by step reflection process and how these may be employed to develop better skills in both teaching and learning system development. The effort so far was limited to single loop learning since the situation did not allow to question the premises. For instance, the curriculum together with the course descriptions, mode of teaching and assessment, the duration, etc., the project from the client side and the working arrangement initially agreed upon were not questioned at all as part of the reflection exercise. They were taken as given. Furthermore, it is also relevant to note that the exercise so far was limited to graduate level. Almost all students at this level had prior exposure to software development and as such had some appreciation of the problems discussed that made them quickly buy into the importance of the approach. This may not be the case with undergraduate students who do not have such exposure as they come directly from high school. Accordingly, a different mechanism may need to be devised for this category.

Taken together, further work is still required to explore innovative ways and means of realizing this mode of teaching in the local setting. Based on such encouraging results, the plan in the future is to introduce more structure into the reflective learning system so as to ensure that the techniques are effectively used for mastering both soft and technical skills. In particular, the development of a flexible framework and of mechanisms to systematize the guidance provided by the tutors, the introduction of a better support structure, the development of training materials and tools, as well as the mode of assessment, etc. will be explored.

Once enriched and tested as such, sharing this experience with others in order to further develop it and integrate it in the curriculum of higher learning institutions needs to be explored. In parallel, how to make institutional level arrangements between university departments and the industry or public services to support such real-life project based teaching needs to be worked out as well.

7.2 Ongoing Experience at Organization B

As we tried to establish in the preceding chapters, the Reflective Steps approach evolved out of extensive experience in system development teaching and practice. Among the projects that contributed to the evolvement of the approach were the first two software development projects at Organization B (refer to Chapter Two for more). As the application of the method and the work on its improvement is an ongoing process, an attempt is made here to briefly report on one such effort. The purpose here is basically to update what was already reported in preceding chapters in connection with the application of aspects of the Reflective Steps approach in a real-life project at Organization B.

(i) Project Design

Guided by the Reflective Steps approach the following activities were carried out with regard to the project under reference. In this connection, it is also relevant to note that what is presented below builds on the case reported earlier in Chapter Two in respect of Organization B.

When the management of Organization B decided to wait no longer for the software under development in the second round project documented in Chapter Two, it was also decided to commission another fresh software project to address its unmet needs. The researcher was approached by the management of Organization B to help in this renewed initiative. In response, a series of assessment type discussions were held between the researcher and the top-management of Organization B basically to reflect on the past experiences and draw lessons to help design the new project. The discussions and the outcomes from the series of meetings were captured in a draft prepared by the author.

The draft was then reviewed by the management and consensus was reached particularly on the scope and mode of software acquisition. Due to the repeated failure of previous attempts to acquire the required software through a custom-building strategy, the management questioned the feasibility of such option and decided to change it (in a sort of double-loop learning). As a result, an acquisition option that involved the purchasing of an off-the-shelf software and customization of same to the needs and requirements of the organization was considered instead. In addition to other decisions made based on the lessons learned, the researcher was also mandated to further develop the project and advise Organization B on the best way to achieve its desired objective through the implementation of the revised strategy in the shortest possible time. An agreement was also made to regularly meet and review progress of the project design activity.

The author started to work on the project design through iterative and evolutionary steps. In particular, consultations were first made with users in the various departments, including those that were involved in the previous attempts. At the time of such consultation, the in-house IT manager had already left the organization, so the discussions were limited to end-users, middle management and professional colleagues. The draft document prepared earlier was then updated using the information obtained during the consultation exercise. Knowledge of the previous projects did help much in this exercise.

The updated version of the draft was then presented to management for review. As part of the updated draft, a proposal was put forward to engage an impartial consultant team to work with the researcher in the implementation of the project. In this setup, the proposal was for the researcher to assume the role of an external expert closely working with the consultant team and the management of Organization B. This was fully endorsed by the management. In the discussion that followed the endorsement, an attempt was made to jointly explore the composition of such an impartial consultant team. It was agreed that Organization B would benefit if such a consultant team was composed of practitioners and academicians or university researchers in the field of software development.

Accordingly, the researcher formed a consultant team composed of:

- six scientists (with experience in teaching and consulting in the areas of software development), two PhD holders, two PhD students, two master's holders,
- three senior postgraduate students,
- three domain experts from Organization B,
- two in-house IT experts (to be employed), and
- the researcher.

In the formation process, each member was independently briefed about the project and was asked if he/she was interested and willing to work on such a project. The researcher then brought together those interested and willing, and formally shared with them the previous project stories and experiences, as well as the expectations of the client from the new project. This was followed by another session, where the researcher further elaborated on ongoing activities by way of reviewing the existing situation and management priorities.

The newly established team was then introduced to the management of Organization B. The preparation (orientation) helped the team to converse well with the management on the planned project. While the team got an opportunity to see the interest and commitment of the management on the one hand, the management felt more comfortable when they saw the familiarity (within such short period time) of the newly established team with the project on the other. After some discussion on the way forward, it was decided that the team develop its own terms of reference for the engagement in close consultation with the management. After a series of planning type reflection sessions with selected members of the management, the group formulated a better picture of the situation and what was expected of the consultant team. As a first step, the team developed an inception report based on the previous draft and findings of the planning type reflection sessions. This was reviewed and enriched by the management of Organization B and key representatives of the major business units to be considered in the first round software development. Once this was done, the team then developed the first version of the project proposal in two consecutive workshops. As part of this proposal, the team decided to adopt the Rational Unified Process (RUP) as a starter approach. This was in fact partly the reason for developing the inception report mentioned above. The team also decided that the elaboration phase be done by the team before publishing the invitation to bid to select the partner for the supply and customization of the required insurance software. Based on these decisions, the team finalized its terms of reference and presented it to the management of Organization B. The proposal was openly discussed and endorsed after making the required modifications. An official agreement was then signed between Organization B and the team to that effect.

Based on the inception report and the terms of reference approved, the team of consultants developed a starter but comprehensive project management plan that defined among others: the project scope, objectives, timetable, work products, work breakdown structure, organization and staffing, etc. An aspect of this plan that is worth noting here relates to the inclusion of the mode of work and communication, together with roles and responsibilities of the team and project governance structure. The project structure included a Steering Committee, that was chaired by the Director General of Organization B, and included senior managers from operation departments, the external expert (the researcher), the project manager and the in-house IT expert (by this time the post of the IT Manager was filled up by direct employment). Reporting to the Steering Committee and directly responsible for the management of the project was a Project Management Group.

Among the roles established at project level were: a Project Leader, a Process Expert, a Communication/Documentation Expert, a Process Owner, Domain Experts, User/Sponsor Representatives and Software Specialists. The Process Expert played the role of a project management process controller who was there to ensure whether the project was carried out in accordance with the methods and procedures agreed upon jointly by the group. The Project Leader played the role of a person who facilitated discussions, reflections and coordination activities among the various sub-teams and between the team and the client. The Communication Expert played the role of a person who ensured that information about the project was shared and timely communicated, deliberations of the reflection workshops were captured and made accessible to members of the project group and other stakeholders. The Project Leader, the Process Expert and the Communication Expert worked together in analyzing and synthesizing the deliberations and outcomes of the reflection workshops conducted at various stages during the actual conduct of the project. In addition, in between reflection workshop sessions, this group met with the external advisor and user representatives to clarify misunderstandings and conflicts that arose during the project work, to update management on the status and progress of the project, etc.

(ii) Business Process Documentation

As part of the elaboration phase, the team worked for about twelve weeks to learn about the application domain and in the process document the business process using use cases and related object-oriented tools per the recommendations of RUP. UML and Rational Rose tools were heavily used for documentation. The researcher worked actively in his capacity as an external expert fully charged with the project design and development supervision. As an action researcher, the researcher established and worked with the consultant group and users to document/develop interpretations of business processes and requirements, in addition to providing overall project leadership (by citing and sharing previous experiences as required). The documentation of the business process redesign prepared in the second project reported previously was also made accessible to the team. Required additional information was collected through observation and from a series of focused group discussions that took place on fortnightly basis in the presence of the project team and users and the monthly meetings of the steering committee.

Three types of workshops were regularly conducted to support the work in the elaboration phase. One series was concerned with building the skills of the consulting team in both technical and social aspects of methods and processes employed. In these workshops, which were conducted on weekly basis, team members shared experiences among themselves on the concepts, techniques and tools of software methods and processes in general and those related to RUP in particular. For these purposes, practical cases from the project (that is, work products developed as part of the project) were used as objects of discussion and reflection. Individual and group assignments were given to prepare and share experiences on selected topics of relevance to the project work. Such group assignment topics included: use and application of certain tools and techniques, comparison of methods, further investigation of problems encountered during the work, etc. As part of the assignment, a member or a group conducted some research (in the form of literature review, experimentation with certain tools, etc.) and made presentations to the group based on the findings. Other members questioned and criticized aspects of the presentation in order to learn about the topic of presentation. These workshops helped members of the consulting team to have a shared understanding and enhanced skills on the use and application of methods and processes in general and RUP in particular.

The other series of workshops concerned understanding the insurance application in general and the business process at Organization B in particular. Likewise, in these workshops (which were conducted on fortnightly basis), presentations followed by group discussions were used as vehicles of understanding and knowledge building. These workshops were attended by employees selected from the various departments in the Head Office and Branch Offices. Initially, the presentations were primarily prepared and made by members of the consulting team (the IT Group). In later sessions, however, user counterparts working as domain experts (the Operation Group) were made to prepare and present cases at the workshops. This did not only help to demonstrate ownership among the domain experts, but also resulted in building confidence among the audience (the user community in particular). The attendance, enthusiasm and active participation of employees were observed to be much better in those workshops where the domain experts made the presentations. Valuable comments, particularly in the areas of workarounds introduced in practice and special processes introduced to attract more customers, were made to enrich knowledge on the application domain and special services provided.

The third series of workshops related to progress review. These workshops were conducted on a monthly basis at the Project Management level and on a quarterly basis at the Steering Committee level. These workshops particularly compared intended plans with actual performance, and invited discussions on the variances. Special attention was paid to analyze causes for both weaknesses and strengths of project performance. Based on the causes established, corrective measures were devised. As required, issues that were not addressable at the Project Management level were brought to the attention of the Steering Committee (this was done based on the issue escalation scheme developed as part of the project plan). Per plan, the reports from the reviews were circulated to all members of the project staff. Copies of the reports were also submitted to the Steering Committee members.

While there were a number of problems encountered (see below), most of them were resolved at the project team level, of course in close consultations with the management of Organization B. Among the factors that negatively affected project performance and were reported to the Steering Committee were the delay in filling the in-house IT Manager position and the work overload on user counterparts. Both problems were resolved through the intervention made by the Director General. The user counterparts were generally very motivated. They worked very actively with the consulting team most of the time. Due to the workload in their regular assignments (operational responsibilities), however, they had to miss out in some of the critical sessions. This created information gaps that at times slowed down work progress somehow. After deliberations at the Steering Committee level on the matter, a decision was made to relieve/excuse them of their regular duties as long as they were needed for the project work.

Working intensively in this manner, the team was able to complete the elaboration phase one month after the original plan. The delay, however, was not received by surprise by stakeholders as they were being informed through the regular monthly progress reviews. As a work product, a comprehensive and a detailed business process documentation was produced using both use cases and object-oriented modeling tools. The outcome of the elaboration phase was also presented to the management.

(iii) Conflict Resolution

Once the business process was documented, the next major activity according to the adjusted software acquisition process was to prepare a bid document and invite potential suppliers. In the discussions held to map out the way forward, major differences featured in the approach to be followed. In particular, the in-house IT manager insisted that the appropriate process model to be followed should be somewhat like the traditional waterfall. The reasons given as justification related to exercising better management control over the procurement process. The manager claimed to base his suggestions on prior exposure (the IT manager used to work for another company before joining Organization B – the manager was also a former student of the researcher). The consulting team, on the other hand, although open for change, did not see the point of switching to the waterfall type process at this stage. Citing the unfavorable experience of the previously failed cases, which followed a similar process, the consulting team insisted that more iterative and incremental approaches based on prototyping be considered for implementation.

The IT manager also recommended to the management that the role of the consultant team be redefined. The recommendation in particular was to separate the activities of the in-house IT unit and the consulting team, where the latter would be checked by the former as to its performance in accordance with revised terms of reference. A series of recommendations was also forwarded, based for the most part on traditional approaches of project control. In addition, requests were made to include more general business applications (such as human resource and fixed asset) in the priority list. Another interesting development (for the researcher) was the insistence of the IT manager to discourage direct user participation in the workshops and in the development activities. Citing his years of experience in the industry and previous involvement in similar software acquisition processes, the IT manager claimed that he was better equipped (than the users) with most of the information required for the process of acquisition. Even where there was a need for additional information, his office should be able to work it out with the user units through formal administrative channels. To cut the story short, the IT manager was insisting on the use and application of traditional approaches.

Although attempts were made at the beginning to facilitate discussions between the consulting team and the in-house IT Manager to resolve the disagreements, this did not succeed. Gradually the disagreements grew into a serious conflict. At this stage, instead of confronting, members of the consulting team opted to withdraw. For one thing, according to them, they were unable to cope with the swift changes being proposed in the strategy without consultation and discussion and thus unable to understand the positions taken by the IT manager. Under the circumstances, they considered him not trustworthy to work with. On the other hand, as long as the IT manager felt he knew what he was doing and was able to convince management about it, the consultant team did not see the need for external consultants. Furthermore, working with the new arrangement did not interest them at all as researchers and students of software engineering.

The conflict had serious impact on the progress of the project. Considerable delays were reported in the various review meetings. At this point, the researcher had to intervene. An extraordinary Steering Committee was called to preside over the matter. After a series of deliberation, the steering committee asked the researcher to make investigation into the case and come up with recommendations to resolve the matter without further loss of time. At the same time the Steering Committee, particularly the Director General, strongly acknowledged and expressed the contribution of the consultant team and that there should be no confusion around the continuation of the engagement. It was also agreed that the delay in the project schedule could be tolerated until the conflict at hand is resolved.

Accordingly, an assessment of the case was made by the researcher at two stages. First, attempts were made to determine the main causes of the problems encountered in the progress of the project. Based on the findings, the next step was to outline corrective measures to be taken in order to expedite the successful implementation of the project.

The researcher, after making a series of discussions with both groups independently and jointly, prepared a report on the findings and the way forward. The report was also reviewed by the consultant team and the IT manager before its submission to the steering committee. Where disagreements were expressed, they were noted in the same report.

Among the major issues identified during the assessment process as prime causes of the problems encountered were:

- Lack of shared understanding of project scope, process model, roles and schedules;
- Lack of effective project leadership;
- Absence of an appropriate change control process;
- Absence of effective communication;
- Lack of openness and honesty about what one does and does not know; and
- Lack of integrity necessary to admit mistakes.

According to assessments made by the researcher, most of the recommendations made by the in-house IT manager were based not so much on the needs of the project but on what the IT manager knew and experienced. On the other hand, the consulting team completely relegated all kinds of communication with Organization B to the IT manager. For instance, the management was not aware of the disagreements until late, and only knew the IT manager's version of the story. As indicated, instead of engaging and confronting, the consultant team opted to withdraw, failing to shoulder the responsibility entrusted to it by the management, ignorant of its shortcomings in organizational competence, etc. It was easily walked over by the more bureaucratically subtle and outspoken IT manager to the extent of putting the project at risk.

On the part of the IT manager, in addition to the remarks made above, after coming so late in the project, the sort of changes proposed without proper consultation and the inflexibility demonstrated, may well be considered inappropriate. What is more, being suspicious of the consulting team whose members were diligent in the pursuit of the project goals before his arrival, and the need to watch over their shoulder to constantly monitor their activities, may also be considered out of place.

The researcher was also to blame on grounds of the conflict of interests. Although the disagreement was noticed relatively early, the researcher let it surface because of the interest to study it. Although disagreements and conflicts were expected between users and developers, these were not particularly expected to manifest at this scale between external consultants and internal technical personnel. So it was an interesting case to look into. And the investigation[18] gave an opportunity for this. However, the researcher is to blame because of the interest to research into the case at the expense of the project (putting the project at risk). With full confidence from the management of Organization B and the respect from the professionals (as all members of the consultant team as well as the IT manager were former students of the researcher) the researcher should have intervened earlier than this.

The report prepared by the researcher was discussed at the steering committee level and corrective measures were taken based on the recommendation. As a result, revisions were made on the project plan including adjustment of approaches, roles and responsibilities. The consulting team and the IT manager were still working together and the project continued slowly but progressively. The invitation to bid was released, bid documents were analyzed and a supplier was selected. At the time of writing this report, contracts were being negotiated with the supplier and in this process an aspect of Reflective Steps was being considered for application. In particular, the proposal with regard to formally establishing a collaborative development team was presented for consideration by the project management team.

[18] The data collected and the analysis still continued and the full and detailed account will be published and shared in due course of time.

CHAPTER EIGHT

8. Conclusion

As part of the researcher's effort over the years (supported by his postgraduate students) to ameliorate the software development situation locally, a research project was initiated in parallel with teaching system development methods. The research reported here is part of this effort. It related to tailoring complementary aspects of suitable software development approaches (methods and processes) with the dual purpose of: enriching the approaches themselves, and developing a context-sensitive methodical approach that would contribute to the improvement of the software development situation in Ethiopia.

The underlying assumption in this research undertaking was that contextualization (to the national context) and then customization (to a project context) of complementary aspects of popular and widely used approaches would enhance the usability and application of methods and processes in the software development effort. This in turn would improve the local software situation and success rates of projects by contributing to the effort to bridge the oversize supply-demand gaps.

Based on extensive review of related literature, years of experience in teaching and practicing software development, and background work done for the purpose of this research, the following strategy was developed and followed to come up with a context sensitive approach.

- Overall assessment of the software development situation in Ethiopia for the purpose of documenting the situation and the identification of factors that characterize the national context in so far as determining suitability or customization of software methods and processes is concerned.
- Closer examination of publicly available and widely used software methods and process models for the purpose of understanding concerns and issues currently being addressed by the frontiers in the industry and identification of those to be considered for contextualization to the local settings.

- Contextualization of selected methods and processes to the national level through modification and incorporation of locally developed techniques and proven practices.
- Customization of the contextualized methods and processes for use in a specific project setting.
- Providing mechanisms through which software processes are established and continuously improved in software companies.
- Building skills and competencies of both practicing software engineers and students studying software engineering in higher learning institutions.

Projects operate within various levels of, often nested, contexts. For our purpose these levels were classified into three: global/industry context, national context, and project context as described below.

8.1 Industry (Global) Context

It has already been recognized among the software engineering community that the software development process can not be fully formalized because it is a process that demands: high social competence and team work, understanding of the use-situation and consideration of organizational embedding. Accordingly, the scientific research interests of the community have gone beyond the formal and mathematical methods provided for in traditional computer science. In particular, in order to address the social aspects of software development, the focus has shifted to exploring the tailoring of approaches developed elsewhere for understanding organizational and human learning and communication, individual and cooperative work. Historically, the efforts to systematize software development approaches brought about the shift from unstructured and chaotic processes to the traditional plan-driven (product-oriented or phase-oriented) structured, linear time-delineated stage models and defined milestones. As of recent, efforts of the software engineering community (to address the limitations of traditional methods particularly in terms of addressing the social aspects) shifted to exploring approaches which are more flexible, iterative, incremental and collaborative.

However, there are still so many open questions and issues to be addressed around methods and processes for software development. Accordingly, work in the area has continued with focus on improvements in existing methods as well as the development of new and better methods.

8.2 National Context

Although encouraging developments were observed in the network infrastructure development component of ICT at the national level, the situation in application and content development in general and software development in particular is not satisfactory. There is very limited capacity in the software area compared to the huge demand that resulted from the effort to realize investment in the infrastructure on the one hand and efforts to introduce best practices in business operations on the other. Most of the software development projects being outsourced are very large with organization-wide and nation-wide implementation scope. Most of the projects concerned custom-building of business applications for specific organizations. What is more, according to the survey results, about 50% of these projects are from the government sector. On the other hand, the software companies are small, staffed with formally trained (with 85% with university degrees in computer science and related fields) but less experienced (about 75% with less than 5 years of experience) personnel.

This demand-supply gap resulted in a situation that was dominated by high project failure rates. The project success situation could be characterized by,

- Unrealized improvements in business efficiency and value as a result of the software introduction;
- substantial budget and time overruns far beyond expected;
- delivery of unfriendly, and poor quality and thus unused software products;
- difficulty on the part of the users to effectively utilize the newly deployed software systems because of inadequate training;
- incomplete documentation and lack of timely and affordable maintenance support, and hence problem of sustainability, etc.

With regard to method use, the local software development practice is dominated by the use of ad-hoc in-house guidelines that involve cyclical requirement gathering and programming/coding techniques. The use of disciplined project management and software methods and process is very low.

> The situation is partly attributable to technical skill deficiency due mainly to lack of appropriate orientation in the field of software engineering in general and lack of relevant training on methods and processes in particular. There are also limitations in the publicly available processes and methods to fully address local realities. What is more, efforts to tailor these methods to address local realities are non-existent. Moreover, there is the absence of formally documented work practices and related procedures and guidelines within software development companies that demand use and application of methods and processes.

According to the survey results: about 49% of the professionals identified absence of guidelines on method use as critical limiting factors to carry out their tasks effectively; about 57% indicated adoption of guidelines and standards as an area that needed urgent intervention; 65% indicated skill upgrading as an area that needed urgent intervention. More over about 71% of the software companies identified the use of standard methods and disciplined project management as most important to produce quality software on time.

In relation to other performance inhabiting factors, the survey results indicated: change in requirements, poor planning and staff turn over, lack of properly defined roles and responsibilities, to account for 54%, 50%, 40%, and 43%, respectively. Communication was almost informal both within the development team (82%) and between the development team and users (70%). About 50% of the respondents identified coordination with the team and communication with users as challenging tasks.

From reflections made on years of software development experiences in the local setting, the most critical success/failure factors in custom-built application software included the following: joint development of the software with domain experts and business process owners; project scoping; organizational communication and coordination to share experiences and build better understanding on projects; overdependence on few highly motivated and capable user counterparts; knowledge of software engineers on the application domain; conscious and controlled management of changes taking place in user organizations and technological platforms.

8.3 Solution Design Considerations

The following table summarizes the findings with regard to the features of the national context (factors to be considered in the design of methodical approaches) together with methodical strategies devised to tackle these factors.

Context level	Factor	Strategy proposed
Industry/global	Changing requirements Use context Organizational embedding	Methods & processes: • Iterative • Incremental • Participatory • Communication and Learning
National	Project type: • Large projects with national level deployment • Custom-built business applications • Part of organizational reform	Method and process contextualization: • Project scoping and step-by-step delivery • Institutionalized collaborative (user + supplier) development team • Integrated business process redesign and software design approach
	Workforce: • Soft skill deficiency among practitioners • Scarcity of qualified and experienced professionals • High staff turnover	• Work-based reflective learning for practitioners • Integrating real-life projects in course delivery for students of higher learning institutions to develop both technical and soft skills
	Passive user participation and low motivation Communication & coordination Sustainability problem Immature software engineering environment: lack of historical data and project stories, absence of experience sharing platforms and forums, small and young software companies, absence of national guidelines/standards on software methods and processes	• Institutionalized collaborative arrangements with incentives • Use design, organizational communication and embedding • Awareness creation on the need for national capacity building programs
Project	Vary from project to project No/zero learning Absence of methods and process guidelines at organizational level Inadequate domain knowledge	Project-based step-by-step and continuous software process improvement based on collaborative reflective learning

Table 7: Summary of contextual features and corresponding strategies proposed

The factors that were considered as features of the national context were those issues that were commonly recurring in almost all projects.

In general, approaches that:

- presuppose a thorough familiarity with work place realities including the extraordinary non-standardized work practices,
- promote collaborative and participative practices,
- provide flexibility in both project planning and software design, and
- try to benefit from new and emerging technologies and best development practices,

were the ones considered suitable for customization.

8.4 Results So Far

Putting together the strategies outlined above, an attempt was made to develop a methodical approach to software development which is responsive to the local context.

The approach was developed by contextualizing the STEPS model originally developed by Floyd et al. (1989). Contextualization of STEPS to the national context resulted in a methodical approach known as Reflective Steps.

Reflective Steps integrates three main software processes: project design (consisting of one or more application software development efforts), application production (concerned with the incremental design and development of an application software), and application use (concerned with the organizational embedding, sustainability and software maintenance). The cycles at project design level include incremental delivery of a project, application by application, where each cycle involves the review of project scope, priority, contract, etc. as developed applications are delivered. The cycles at application production level include incremental delivery of an application, increment by increment, where each cycle involves the revision of processes, plans and products upon delivering an increment. The cycle at application use level includes embedding operational versions of the increments delivered, version by version. Each cycle involves installation of an operational version of the software increments developed, related revision of work procedures, training, troubleshooting of software errors and determination of revision requests on delivering an integrated version of increments developed up to that point.

The contextualized process model emphasises the explicit treatment of such important activities to the local situation as consensus building, make-or-buy decision, project portfolio management, business process redesign, among others. It involves the incorporation of techniques developed by the author and proven practices in the area of software project design and management methods (through selection and customization of techniques and tools from such general project management standards as PMBOK and Prince-2). It also incorporates a home-grown innovative and collaborative approach in the area of business process redesign for software development.

For project level customization, continuous project-based process improvement through collaborative reflection learning based on project experience is proposed. To this end, drawing from organizational and individual learning theories, a multilevel learning model, consisting of single-loop learning and double-loop learning, was developed. Each learning cycle involves action-reflection-improvement (or reflection-in-action, reflection-on-action and reflection-for-action), where the reflection in case of design, for instance, involves design-critique-reflection. While the single-loop is concerned with taking corrective measures at the tactical level without the need to revisit premises (goals, processes, etc.), double-loop learning is concerned with taking corrective measures at strategic level which involves questioning and revisiting the premises and basic assumptions, and redesign of processes. In addition to process improvement, the learning model developed is also suggested for use for managing changes to be introduced in the product and in the project management.

From the experience of applying Reflective Steps in real-life project environments, a Reflective Steps Workshop technique was developed to guide practice.

To address skill deficiencies of practitioners and students, collaborative reflective learning based on project experiences is proposed. To facilitate this, in the case of students, integration of real-life projects in the course delivery is suggested.

Encouraging results were obtained in the field experiments conducted in real-life problem situation, both in software development practice and in teaching at postgraduate studies. Based on performance levels demonstrated, there is enough evidence to conclude that the proposed method has the potential to improve teaching and practicing software development in the local setting.

8.5 Overall Observation

Software development is a collaborative work where the product is developed by a jelled development team composed of:

- technical members with knowledge of software development (software engineers, method experts, etc.),
- members with knowledge of the application domain (key user representatives, business process experts, etc.), and
- members with knowledge of project or change management (external experts).

To develop a usable system, there should be meaningful user participation in all phases of the development process, where the users work together with designers to build systems that fit their needs.

If software productivity and quality are to be improved, there is a need to devise ways and means of facilitating continuous sharing and integration of domain knowledge, software process knowledge, project progress knowledge, knowledge in mutual understanding, amongst the project staff and concerned stakeholders. In the process, practitioners must learn habits for inquiry, communication and problem solving through collaborative reflection. At the same time they should develop technical knowledge of the software engineering discipline.

No software development project exists in isolation; it operates within some context. The context within which a project operates may be repeatable but may never be totally the same. The features of the context are factors in the method selection, tailoring and use. It is not possible to talk about suitable approaches independent of context and experience-based learning.

Project stories help us make sense out of our experience. Stories about past and present projects, properly documented, are instruments for design and learning. Such stories help in creating a shared understanding and meaning with others who want to join in the dance of discovering how to design, develop and use software within a given context.

The step by step approach to software development enables the development team to demonstrate results earlier and faster. It also provides an opportunity to continuously refine plan, process, and design for the next increment.

Suitable processes and methods for a project are developed in the course of developing the software itself by using a contextualized approach as a starter and then customizing the contextualized approach in a step-by-step improvisation process through collaborative reflective learning based on project experience.

Process improvement and product development exist in a co-creative relationship with one another. The process affects the quality of the product; and the product developed is used to evaluate the contribution of the process, resulting in the improvisation of the process itself to help develop better quality products in subsequent steps.

Beyond the lecture-based methods that use textbook cases and simulations, greater focus needs to be given to integrate real-life problem scenarios into the curriculum in order to help students establish connections between the discipline and the world beyond the classroom. In the process, students get an opportunity to build better skills in organizational competence, communication, collaboration, team-working abilities, and creative problem-solving.

8.6 Future Work

With specific reference to the research questions outlined at the outset and the achievements reported, the following generic recommendations are made for consideration in the times ahead.

- With regard to further developing the proposed approach, more practical experimentation and testing are required to concretize, operationalize and enrich the proposed approach. Continuous assessments need to be done on the extent to which the use of this approach practically improves the existing software development situation, particularly in realizing such operational benefits as lower costs, timely implementation, rise in quality, lower defect rates, flexibility to change, and the ability to leverage new technical or business information. Further work in the direction of project portfolio analysis for large project is critical.
- The situation assessment instrument developed for the purpose of this study needs to be further developed in the form of standard instruments and guidelines that may be adopted at national level for the purpose of assessing the capabilities of local software firms, with a view to improve their competencies. Efforts in this direction may be integrated with the contextualization of CMMI or related techniques and tools to local settings.
- Further work on the cultural dimension of collaborative approaches, including motivational factors in the local context, is required for successful application and use of Reflective Steps.
- At the technical level, mechanisms to extend pair programming principles commonly practiced in agile methods to those used in the case studies reported (i.e., to support the work of two software engineers and one domain expert); mechanisms to integrate the role of a facilitator, scribe and process expert in project team establishment; and the tailoring of the 'function-mechanism' framework proposed for other applications by Miyake (1986) to supplement the object oriented techniques of decomposition are all worth exploring.
- Efforts to integrate real-life projects into the teaching of software development will no doubt help in addressing the skill deficiencies of students of higher

learning institutions, as well as preparing them to work place realities. Successful implementation of this in a sustained manner would, however, require further work in the areas of reorientation in the perception and practice of software development and software engineering, systematizing and institutionalizing the effort so far through formal revisions of software engineering curriculum content and content delivery (how the content is taught).

In addition, at the national level, awareness creation programs on the software development situation, platforms for sharing of project stories (information, experiences and learning), national guidelines and standards for software methods and processes, and related capacity building programs need to be developed and facilitated to support the smooth implementation and successful utilization of methodical approaches developed in this and related works.

REFERENCES

Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J. 2002. *Agile software development methods: review and analysis.* On-line. Available from Internet, http://virtual.vtt.fi/inf/pdf/publications/2002/P478.pdf., accessed 19 October 2007.

Ahmed K. 2006. *An assessment of user participation practice on customer care and billing project of ETC with special reference to participatory design methodology.* M. Sc. Thesis. College of Telecommunications and Information Technology. Addis Ababa.

Arent, J., Iversen, J. H, Andersen, C.V, and Bang, S. 2000. Project assessments: supporting commitment, participation, and learning in Software Process Improvement. *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences. Hawaii*. 4-7.

Argyris, C. 1994. *On organizational learning*. Malden: Blackwell Publishers.

Argyris, C. and **Schön, D.A**. 1996. *Organizational learning II: theory, method and practice*. Massachusetts: Addison-Wesley.

Argyris, C. and **Schön, D.A**. 1978. *Organizational learning: a theory of action perspective*. Massachusetts: Addison-Wesley.

Austin, R. and **Paulish, D.** 1994. A survey of commonly applied methods for software process improvement. *Technical Report*. CMU/SEI-93-TR-027. US Department of Commerce: Springfield.

Avison, D. and **Fitzgerald, G.** 1995. *Information systems development methodologies, techniques and tools*. London: McGraw-Hill.

Avison, D. Fitzgerald G., and Powell, P. 2001. Reflections on information systems practice, education and research: 10 years of the information systems journal. *Information Systems Journal.* 11 (1): 3-22.

Bale, L.S. 2007. *Gregory Bateson's theory of mind: practical application to pedagogy*. On-line. Available from Internet, http://www.narberthpa.com/Bale/lsbale dop/learn.htm, accessed 19 October 2007.

Baker, M.J. 1999. Argumentation and constructive interaction. In. *Foundations of Argumentative Text Processing*. J. Andriessen and P. Coier (eds.). Amsterdam: University of Amsterdam Press.

Barnden, A. and **Darke, P.** 2000. A comparison of SSM with an organisational learning model. In: *Proceedings of the International Conference on Systems Thinking in Management (ICSTM2000),* 89-94. G. Altmann *(ed.).*Australia :Geelong.

Barrett, L. and **Lehtonen,K**. 2004. *Managing a product development team: part ii – growing the team*. On-line. Available from Internet, http://www.dau.mil/ pubs/dam/05_06_2004/bar-mj04.pdf, accessed 8 October 2007.

Basili, V. R. and **Caldiera, G**. 1994. Experience Factory. In: *Encyclopedia of Software Engineering*. (ed.). Marciniak, J. J. 469-476. John Wiley & Sons, Inc.

Baskerville, R.L. 1999. Investigating information systems with action research. *Association for Information Systems*, Atlanta 2(3).

Bateson, G. 2000. *Steps to ecology of mind: collected essays in anthropology, psychiatry, evolution, and epistemology*. Chicago: University of Chicago Press.

Bateson, G. 1979. *Mind and nature: a Necessary unity,* Bantam Books. *Advances in Systems Theory, Complexity, and the Human Sciences*. Hampton Press**.**

Beck, K. 2004. *eXtreme Programming Explained: Embrace Change*. 2nd Ed. England: Addison-Wesley.

Birk, A. and Pfahl, D. 2002. A systems perspective on software process improvement. *Lecture Notes in Computer Science*. 2559:4-18. Heidelberg: Springer Berlin.

Bjerknes, G. 1993. Some PD advice. *ACM*. 36(6): 39.

Bjerknes, G.1992. Dialectical reflection in information systems development. *Scandinavian Journal of Information Systems*. 3: 55-77.

Bjerknes, G. **Dahlbom, B., and Al, L.E.** (eds.). 1990. *Organizational Competence in system Development*. Lund: Student-litteratur.

Boahane, M. 1999. Information system development methodologies: are you being served? In: *the Proceeding of 10th Australian Conference on Information Systems*. Wellington.

Boehm, B. W. 2002. Get ready for agile methods, with care. *Computer.* 35(1): 64-69. IEEE Computer Society Press: Los Alamitos.

Boehm, B. W. 1989. Theory W Software project management principles and examples. *IEEE Transactions on Software Engineering***.** 15(7): 902-916. Piscataway: IEEE Press.

Boehm, B. W. 1988. A spiral model of software development and enhancement. *IEEE* 21(5): 61-72.

Boehm, B. and **Turner, R.** 2003. Using risk to balance agile and plan-driven methods. *Computers.* 36(6): 57-66. Los Alamitos: IEEE Computer Society Press.

Boehm, B. W., **Egyed, A., Kwan, J., Port, D., and Shah, A.** 1998. U*sing the WinWin spiral model: a case study.* Los Alamitos: IEEE Computer Society Press.

Boehm, B. W. and **Bose, P.** 1994. A collaborative spiral software process model based on theory**.** In: *Proceedings of the 3rd International Conference on Software Process. Reston, USA.*

Boehm, B. W., Bose, P., Horowitz, E., and Lee, M. 1994. Software Requirements As Negotiated Win Conditions. *Proceedings of the First International conference on Software Process***.** IEEE**.**

Boland, R. J. 1978. The process and product of system design. *Management scienc*e. 24(9): 887-898.

Bond, C. and **Kirkham, S.** 1999. Contrasting the application of soft systems methodology and reflective practices to the development of organizational knowledge and learning: a review of two cases in the UK national health services. *Proceedings of the 1999 ACM SIGCPR conference on Computer personnel research*. 242-252. New Orleans.

Bostrom, R. P. and **Heinen, J. S.** 1977. *MIS problems and failures: a socio-technical perspective. part I: The Causes.* University of Minnesota: Management Information Systems Research Center.

Boud, D. 2001. Using journal writing to enhance reflective practice. In: English, *Promoting Journal Writing in Adult Education.* New Directions in Adult and Continuing Education*. 90: 9-18*. Gillen, M. A. (ed.). San Francisco: Jossey-Bass.

Boud, D. Keogh, R. and Walker, D. 1994. Introduction-what is reflection in learning?. In: *Reflection: turning experience into learning.* 7-17. D Boud, R. Keogh and D. Walker. (eds.). New York: Nichols Publishing Company.

Boud, D., Keogh, R., and Walker, D. (eds.). 1985. *Reflection: turning experience into learning*. London: Kogan Page.

Budgen, D. 1999. Software Design Methods: Life Belt or Leg Iron? *IEEE Software*.16(5): 133-135. IEEE.

Burgess, Y. K. C. and **Conklin, J.** 1990**.** Report on a development project use of an issue-based information system. In: *Proceedings of CSCW'90.* 105-118. New York: ACM..

Caelen, J. and **Jambon, F.** 2006. A Platform for the Participatory Design. *An International Symposium.* France.

Carmel, E., Whitaker, R.D., and George, J.F. 1993. PD and joint application design: a transatlantic comparison. *Communication of the ACM.* 36(4): 40-48.

Caroll, J. M. 1995. *Scenario-based design: Envisioning work and technology in system development.* New York: John Wiley and Sons.

Checkland, P. and **Scholes, J.** 1999. *Soft systems methodology in action*. New York: John Wiley and Sons.

Christel, M. G. and **Kang, K. C.** 1992. *Issues in Requirements Elicitation*. On-line. Available from Internet, http://stinet.dtic.mil/oai/oai?verb=getRecord& metadataPrefix= html&identifier=ADA258932, accessed 21 October, 2007.

Clement, A. and **Van den Besselaar, P.** 1993. *A retrospective look at PD projects.* New York: Communication of the ACM, ACM Press.

Cockburn, A. 2006. *Agile software development: the cooperative game*. 2nd Edition. The Agile Software Development Series.

Constantine, L L. 2002. P*rocess agility and software usability: toward lightweight usage-centered design*. Available from Internet, http://www.uml.org.cn/jiaohu /pdf/agiledesign.pdf, accessed 12 August, 2007.

Constantine, L. L. 1989. OO and S methods: towards integration. *American Programmer.* 2 (7/8).

Curtis, B., Krasner, H., and lscoe, N. 1988. A field study of the software design process for large systems. *Communication of the ACM*. 31(11): 1268 – 1287.

Curtis, B., Kellner, M.I. and Over, J. 1992. Process modeling. *Communications of the ACM*. 35(9): 75-90.

Dada, D. 2006. The Failure of E-Government in Developing Countries: a Literature Review. *EJISDC.* 26(7):1-10.

Dahms, M. and Faust-Ramos, E. 2002. Development from Within: Community Development, Gender and ICTs. In: *Feminist Challenges in the Information Age, Leske + Budrich, Opladen.* 267-286. C. Floyd, G. Kelkar, S. Klein-Franke, C. Kramarae, and C. Limpangog (eds). International Women's University.

Dalcher, D. and **Drevin, L.** 2003. *Learning from information system failure by using narrative and anti-narrative methods*. South African Institute for Computer Science and Information Technologies.

Danielsen, T., **Pankoke-Babatz, U., Prinz, W., Patel, A., Pays, P., Smaaland, K., and Speth, R.** 1986. The amigo project – advanced group communication model for computer-based communications environment. *Proceedings of the 1986 ACM conference on Computer-supported cooperative work.* 115-142. New York: ACM Press.

Debrabander, B. and **Edstrom, A**. 1977. Successful information system development projects. *Management science*. 24(2).

Dewey, J. 1933. How We Think: A restatement of the relation of reflective thinking to the educative Process. Lexington, Massachusetts, D C Heath.

Diwan, A., Waite, W. M., and Jackson M.H. 2002. *An Infrastructure for Teaching Skills for Group Decision Making and Problem Solving in Programming Projects*. New York: ACM Press.

Dohlbom, B. and **Mathiassen, L.** 1993. *Computers in context: the philosophy and practice of system design*. Blackwell Publishing Limited.

Drohan, S., Stapleton, L., and Stack, A. *2006.* P*roblem solving skills in information systems development curricula.* Online. Available from Internet, http://www.aishe.org/events/2005-2006/conf2006/proceedings/paper-02.doc. accessed 21 September, 2007

Enayati, J. 2002. The research: effective communication and decision-making in diverse groups. A Chapter in Hemmati, Minu. *Multi-Stakeholder Processes for Governance and Sustainability - Beyond Deadlock and Conflict*. London: Earthscan.

Faraj, S and **Sproull, L.** 2000. Coordinating expertise in software development teams. *management science.* 46(12):1554-1568.

Figueroa, M. E., Kincaid D. L., Rani, M., and Lewis, G. 2002. *Communication for social change: an integrated model for measuring the process and its outcomes.* The Rockefeller Foundation.

Fleck, R. 2003. Supporting reflection and learning with new technology. In: *Human Centred Technology Workshop 2003*. 50-52. University of Sussex at Brighton.

Flores, F., Graves, M., Hartfield, B., and Winograd, T. 1988. Computer systems and design of organizational interaction. *ACM Transactions on Information Systems.* 6(2):153-172.

Fitzgerald, B. 1996. Formalized systems development methodologies: a critical perspective. *Information Systems Journal*. 6(1): 3-23.

Fitzgerald, B., Russo, N. L., and O'Kane, T. (2003), "Software development method tailoring at Motorola", *Communications of the ACM.* 46(4): 65-70.

Flood, R. and **Romm, N.** 1996. *Diversity Management: Triple Loop Learning*. John Wiley & Sons Ltd.

Floyd, C. 2005. Being critical in, on or around computing? *Communication of the ACM*. 207-211. ACM Press.

Floyd, C. 1992. Human Questions in Computer Science. In: *Software Development and Reality Construction.* 15-27. C. Floyd, H. Züllighoven, R. Budde and R. Keil-Slawik. (eds.). Springer-Verlag.

Floyd, C. 1992. Software development as reality construction. In: *Software Development and Reality Construction*. 86-100. C. Floyd, H. Züllighoven, R. Budde and R. Keil-Slawik. (eds.). Springer-Verlag.

Floyd, C., Reisin, F.-M., and Schmidt, G. 1989. STEPS to software development with users. In *ESEC'89: 2nd European Software Engineering Conference, Lecture Notes in Computer Science*. 387: 48-64. Ghezzi, C. and McDermid, J.A. (eds.). Berlin: Springer-Verlag.

Floyd, C. 1987. Outline of a paradigm change in software engineering. In *Computers and Democracy - a Scandinavian Challenge*.191-210.

Floyd, C. 1986. A comparative evaluation of system development methods, In *proceeding of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems.* 19-54. Amsterdam: North-Holland Publishing Co.

Gasston, J. and **Halloran, P**. 1999. Continuous software process improvement requires organizational learning: an Australian case study. *Software Quality Journal.* 8(1): 37-51.

Getahun G. 2006. *Adapting requirements elicitation methodology framework by drawing lessons from customer care and billing project at ETC.* M. Sc. Thesis. College of Telecommunications and Information Technology, Addis Ababa.

Goguen, J. 1992. The Dry and the wet. In *Proceedings of IFIP Working Group 8.1 Conference*.1-17.

Green, T. R. G., Payne, S. J., and Van der Veer, G. C. (eds.). 1983. *Psychology of computer use*. London: Academic Press.

Green, D. and **DiCaterion, A**. 1998. Survey of system development process models, *CTG.MFA – 003*: Penn State.

Greenbaum, J and **Mathiassen, L.**1990. Zen and the art of teaching systems development. In *Computers and Society, ACM*, 20(1): 26-30.

Gregor, S and **Jones, D**. 2003. The formulation of design theories for information systems. In *Constructing the infrastructure for the knowledge economy: Methods and tools, theory and practice.* 83-93 Henry Linger, W. Gregory Wojtkowski, and Joze Zupancic (eds.). New York: Kluwer Academic.

Hall, P. and **Fernandez-Ramil, J.** 2007. *Managing the software enterprise: software engineering and information systems in context.* (1st ed.). Int. Cengage Bussness Press.

Halloran, P. 1999. *Organizational learning from the perspective of a software process assessment & improvement program.* Washington: IEEE Computer Society.

Hansen, B., Rose, J., and Tjørnehøj, G. 2004. Prescription, description, reflection: the shape of the software process improvement field. *UK Association of Information Systems Conference*. Glasgow.

Hart, D.N. and **Gregor, S. D**. (eds.). 2004. I*nformation systems foundations: constructing and criticizing.* Canberra:ANU Press

Hazzan, O. and **Dubinsky, Y**. 2005. *Social Perspective of Software Development Methods: The Case of the Prisoner Dilemma and Extreme Programming.* On-line. Available from Internet, http://edu.technion.ac.il/ Courses/cs_methods/ eXtreme Programming/XP_Papers/XP2005Hazzan&Dubinsky_Social_Theories&XP.pdf. accessed 11 July, 2007.

Heeks, R. 2003. *Most eGovernment-for-Development Projects Fail: How Can Risks be Reduced?* Institute for Development Policy and Management. University of Manchester.

Heeks, R. 2002. *Failure, success and improvisation of information systems projects in developing countries.* Institute for Development Policy and Management, University of Manchester.

Heeks R. 1999. The Tyranny of participation in information systems: learning from development projects. *Working paper no. 4 in Development Informatics series.* University of Manchester.

Heiskanen, A. 1995. Reflecting over a practice: framing issues for scholar understanding. *Journal of Information Technology & People.* 8(4): 3-18.

Henry, C. L. 1981. *Implementation: The key to successful information systems.* New York: Columbia University Press.

Hirschheim, R. and **Klein, H. K.** 1994. Realizing emancipatory principles in information systems development: the case for ethics. *Society for Information Management and the Management Information Systems Research Center, Minneapolis*. 18(1): 83-109.

Holtzblatt, K. and **Beyer, H. R.** 1993. Making customer-centered design work for teams. *ACM.* 6(10): 92-103.

Hunt, A. and **Thomas, D**. 1999. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley.

Jayaratna, N. 1994. *Understanding and evaluating methodologies, NIMSAD: A systemic Framework.* Maidenhead. McGraw-Hill.

Jepsen, L. O., Mathiassen, L., and Nielsen, P. A. 1998. Back to thinking mode – diaries as a medium for effective management of information systems development. In *Behaviour and Information Technology.* 8(3): 207-217.

Jepsen, L. O., Mathiassen, L., and Nielsen, P. A. 1998. Using diaries. A chapter in *Reflective systems development.* Mathiassen*, L.* (ed.) *Vol 1& 2.*

Jones, P. H. 1997. *Handbook of team design: a practitioner's guide to team system development*. McGraw-Hill.

Kakabadse, N. K. and **Kakabadse, A**. 2003. Developing reflexive practitioners through collaborative inquiry: a case study of the UK civil service. *International Review of Administrative Sciences*. 69 (3): 365-383.

Keil-Slawik, R., 1992. Artifacts in Software Design, *In Software Development and Reality Construction*. 168-188. C. Floyd, H. Züllighoven, R. Budde and R. Keil-Slawik. (eds.). Springer-Verlag.

Kettunen, P. and **Laanti, M.** 2005. How to steer embedded software project: tactics for selecting the software process model. *Information & Software Technology* 47(9):587-608.

Kimaro, H. C. and **Nhampossa, J. L**. 2005. Analyzing the problem of unsustainable Health Information Systems in less-developed economies: Case studies from Tanzania and Mozambique. *Information Technology for Development*. 11(3):273-298.

Klischewski, R. 2004. Information Integration or Process Integration? How to Achieve Interoperability in Administration, In Proceedings of EGOV (ed.). Traunmüller, R. Springer, LNCS # 3183, Berlin. 57-65.

Kolb, D.A.1984. *Experiential learning: experience as the source of learning and development.* New Jersey: Prentice Hall Inc.

Korpela, M., Soriyan, H.A., and Olufokunbi, K.C. 2001. Activity analysis as a method for information systems development. *Scandinavian Journal of Information Systems.* 12(1-2): 191-210.

Korpela M., Soriyan, H.A., Olufokunbi, K.C., and Mursu, A. 1998. Blueprint for an African system development methodology: an action research project in the health sector. In *Implementation and evaluation of information systems in developing countries.* Vienna: IFIP.

Krabbel, A., **Wetzel, I., and Züllighoven, H.** 1997. On the inevitability intertwining of analysis and design: developing systems for complex cooperations. In proceedings of *DIS'97 Designing Interactive Systems: Processes, Practices, Methods and Techniques.* 205-213. G. Van der Veer, A. Henderson, S. Coles. (eds.). Amsterdam, The Netherlands.

Krabbel, A., Wetzel, I., and Ratuski, S. 1996. Participation of heterogeneous user groups: providing an integrated hospital information system, In *PDC'96 Proceedings of the Participatory Design Conference.* 241-250. J. Blomberg, F. Kensing, E. Dykstra-Erickson (eds.) Cambridge, Massachusetts, USA. 241-250.

Kruchten, P. 2000. *The Rational Unified Process. An Introduction*. England: Addison-Wesley.

Kunda, D. and Brooks, L. 2007. C*omponent-based software engineering for developing countries: promises and possibilities*. On-line. Available from Internet,http://www-users.cs.york.ac.uk/~kimble/research/DC-paper.pdf. accessed 20 October, 2007.

Law, E. 2007 *Reflective design practices in human computer interaction and software engineering*. On-line. Available from Internet, http://www.ics.uci.edu/~redmiles/chiworkshop/papers/Law.pdf, accessed 21 September 2007.

Levina, N. 2005. Collaborating on multi-party information systems development projects: a collective reflection-in-action view. *Information Systems Research.* 16 (2): 109-130.

Livingstone, D. and **Lynch, K.** 2000. *Group project work and student-centered active learning: two different experiences*. Carfax Publishing, Taylor & Frances Inc.

Loy, P. H. 1990. A comparison of object-oriented and structured development methods. *ACM.* 15(1): 44-48.

Lubelczyk, J. and **Parra,** A. 2000**.** Managing the Software Development Process, On-line. Available from Internet, http://www.adass.org/adass/proceedings /adass99/O5-01/, accessed 20 Sept. 2007.

Lynch, M. 2005. *The Design of journals used for reflection.* Master's Thesis, Elton Mayo School of Management. University of South Australia: Faculty of Business and Management.

Mamykina, L., **Candy, L., and Edmonds, E.** 2002. "Collaborative Creativity" , *Communications of the ACM. Special Section on Creativity and Interface*. 45(10): 96-99.

Mathiassen, L. (ed.) 1998. *Reflective systems development. Vol 1& 2.*

Mathiassen, L. and **Purao, S**. 2002. Educating reflective systems developers, *Information Systems Journal*. 12: 81-102.

Mathiassen, L. and Nielsen, P. A. 1990. Surfacing organization competence. Soft systems and hard contradictions. In. *Organization Competence in system Development.* 191-210. G. Bjerknes, B. Dahlbom, and L.E. Al. (eds). Lund:.Student-litteratur,

Mathiassen, L. and **Nielsen, P. A**. 1989. Soft systems and hard contradictions approaching the reality of information systems. *Journal of Applied Systems Analysis*.16:75-88.

McDermid, J. 1993. *Software engineer's reference book.* CRC Press.

Mitev, N. 2000. Toward social constructivist understanding of is success and failure: introducing a new computerized reservation system. *Association for Information Systems.* Atlanta.

Mittelmann, A. 2000: Measuring soft facts in software development. in p*roceedings of* IDIMT-2000 8th interdisciplinary information management talks. 267-277. Hofer, S.; Beneder, M. (eds.). Schriftenreihe Informatik, Band 3, Universitätsverlag Rudolf Trauner, Linz.

Miyake, N. 1986. Constructive Interaction and the Iterative Process of Understanding. *Cognitive Science*. 10(2): 151-177**.**

Moon, F. A. 2000. R*eflection in learning and professional development: theory and practice*. Routledge Falmer.

Mumford, E. 1983. D*esigning human systems: the ETHICS method.* Manchester: United Kingdom.

Mursu, A., Soriyan, H.A., Olufokunbi, K.C., and Korpela, M. 2000. Information system development in developing countries: theoretical analysis of special requirements in Nigeria and Africa. *Proceeding of the 33rd Hawaii International Conference on Systems Sciences.* 185. R.H. Sprague RH Jr. (ed.). Los Alamitos, CA: IEEE Computer Society.

Nakakoji, K. and **Fischer, G.** 1995. Intertwining knowledge delivery and elicitation: a process model for human-computer collaboration in design. *Knowl.-Based Syst.* 8(2-3): 94-104.

Naur P. 1983. Program development studies based on diaries. Green, T., Payne, S., & van der Veer, G. *Psychology of Computer Use*. 159--170. Academic Press.

Naur, P. 1972. An experiment in program development. *BIT*.12: 347–365.

Norman, D. A. 1993. *Turn Signals are the Facial Expressions of Automobiles.* Cambridge: Basic Books.

Orlikowski, W.J. and **Baroudi, J.J.** 1991. Studying information technology in organizations: research approaches and assumptions. *Information Systems Research.* 2 (1): 1-28.

Olle, T. W., Sol, H.G., and Verrijn-Stuart, Alex A. (eds.). 1986. Information Systems Design Methodologies: Improving the Practice. *Proceedings of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the Practice, Noordwijkerhout,* The Netherlands, 5-7. (CRIS '86). North-Holland.

Olle, T. W. Sol, H.G., MacDonald, I.G. 1988. *Information systems methodologies: a framework for understanding*. England:Addison-Wesley.

Osguthorpe, R. T. 1999. *The role of collaborative reflection in developing a culture of inquiry in a school-university partnership: a U.S. perspective.* On-line. Available from Internet, http://www.eric.ed.gov/ERICDocs/ data/ericdocs2sql/ content_ storage _01/0000019b /80/16/52/96.pdf, accessed 22 October, 2007.

Pfleeger, S.L. 1998. *Software Engineering: Theory and Practice*. New Jersey: Prentice-Hall.

Polanyi, M. 1967. *The Tacit Dimension*. New York: Doubleday and Co.

Pomberger, G. 2007, S*oftware engineering education – adjusting our sails.* Power point presentation. International conference on the Opening of Doctoral Program in IT. Adds Ababa University. Addis Ababa, Ethiopia.

Pomberger, G. 2006. Boehm's Spiral Model Revisited. In *Wirtschaftsinformatik – Schluessel zum Unternehmenserfolg.* ed. K. Fink, C. Ploder. Deutscher Universitaetsverlag, Wiesbaden.

Pomberger, G. and Blaschek, G. 1996. *Object-Orientation and Prototyping in Software Engineering*; New Jersey: Prentice Hall.

Pourkomeylian, P. 2002. *Software Practice Improvement.* Doctoral Dissertation. Department of Informatics, Göteborg University. Sweden: Göteborg.

Pressman, R.S. 2003. S*oftware engineering: a practitioner's approach.* London: McGraw-Hill.

Raelin, J. A. 2001. Public reflection as the basis of learning. *Management Learning*, 32(1):11-30.

Raelin, J. A. 1997. A Model of work-based learning, *Organization Science.* 8(6): 563-578.

Rahel K. 2004. S*oftware development assessment in Ethiopia.* Masters Thesis. Department of Computer Science, Addis Ababa University.

Ramesh, B. and **Dhar,V.** 1992. Supporting systems d*evelopment by capturing deliberations during requirements engineering,* 18(6). Piscataway: IEEE Press.

Rauterberg, M. and **Strohm, O.** 1992. Work organization and software development. *Annual Review of Automatic Programming*. 16 (2):121-128.

Rees, D. 2007. *Integrating the "Hard" and "Soft" Sides of Systems Thinking – A Case Study in New Zealand Local Government*. On-line. Available from Internet, http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-72/084%20Rees%20HardSoft.pdf. accessed , 19 October 2007.

Riehle, D. 2000. A *comparison of the value systems of adaptive software development and extreme programming: how methodologies may learn from each other*. On-line. Available from Internet, http://www.riehle.org/computer-science/research/2000/xp-2000.pdf, accessed 19 October, 2007.

Rittel, H.J. and Webber, M. M. 1984. "Planning Problems Are Wicked Problems," *In Developments in Design Methodology*. Ed. N. Cross. 135–144. New York: John Wiley & Sons.

Robey, D. and **Farrow, D.** 1982. User involvement in information system development: a conflict model and empirical test. *Management Science.* 28(1): 73-85.

Rönkkö, Kari. 2005. *Making methods work in software engineering: method deployment - as a social achievement.* PhD Dissertation. Blekinge Institute of Technology.

Salo, O. 2006. *Enabling Software Process Improvement in Agile Software Development Teams and Organizations*. Faculty of Science, University of Oulu.

Sawyer, S. and **Guinan, P. J.** 1998. Software development: processes and performance, *IBM Systems Journal*, 37(4): 552-569.

Scacchi, W. 2002. *Process Modles in Software Engineering*. CiteSeer, IST.

Schön, D. 1987. *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*. San Francisco, CA: Jossey-Bass.

Schön, D. 1983. *Reflective practitioner: How professionals think in action***.**1st ed**.** Cambridge: Basic Books.

Shaw, M. 1990. Prospects for an engineering discipline of software. *IEEE Software*. 7(6): 15 – 24.

Siau, K. and Rossi, M. 1998. Evaluation of Information Modeling Methods – a Review, HICSS (5): 314-322

Sirak G.Y. 1988. *Survey of the Development of Computer Use in Ethiopia.* Addis Ababa. Ethiopia

Sommerville, I. 1996. *Software Engineering*. 5th ed. England: Addison-Wesley.

Sørensen, E. K. 1999. Intellectual amplification through reflection and didactic change in distributed collaborative learning. *International Society of Learning Science.* (71).

Tan, M. 1994. Establishing mutual understanding in systems design: an empirical study. *Journal of Management Information Systems*. 10(4):159 – 182.

Taylor, B. 2004. Technical, practical and emancipatory reflection for practising holistically. *Journal of Holistic Nursing* 22(1): 73-84.

Taylor, B. and **Nurs, J. H**. 2004. Technical, Practical, and Emancipatory Reflection for Practicing Holistically. *Journal of Holistic Nursing.* 22(1): 73-84.

Taylor, C. 2006. Narrating significant experience: reflective accounts and the production of (self) knowledge. *British Journal of Social Work.* 36(2):189-206.

Teferi K. 1994, Information Technology in Ethiopia. In *Information Technology in Selected Countries.* Drew, E.P. and Foster, F.G. (eds.). The United Nations University, Tokyo, Japan

Turner, J., Mavin, S., and Minocha, S. 2006. We will teach you the steps but you will never learn to dance. *The Learning Organization: an international journal.* 13 (4): 398-412.

Takeuchi, H. and **Nonaka, I.** 1986. The new product development game. *Harvard Business review*.

Thomas, M. and **McGarry, F**. 1994. Top-down vs. Bottom-up Process Improvement. *Software IEEE*. 11(4): 12-13.

Thomas, E. 1996**.** Design as Story Telling. Interactions. *The Guide to Computing Literature*. **3(4): 30-35.**

Vygotsky, L.S. 1978. *Mind in Society: the development of higher psychological processes.* Cambridge Ma: Harvard University Press

Wieringa, R. 1998. A survey of structured and object-oriented software specification methods and techniques. *ACM Press.* 30(4):459-527.

Winogard, T. ed. 1996. *Bringing design to software*. England: Addison-Wesley.

Waema, T. M. 1996. Implementation of Information Technology projects and economic development: issues, problems and strategies. In *Global information technology and socio-economic development.* (ed.). Odedra-Straub. New Hampshire: Ivy League. 106-115.

Wetzel, I. 2001**.** Information System Development with Anticipation of Change Focusing on Professional Bureaucracies. *Proceeding of Huwai's International Conference on Systems Sciences, HICCS-34.* Maui.

World Bank, *World Development Report*. On-line. Available from Internet, http://www.wrldbnk.org/wrd/wrd98/index.htm, (1998/99), accessed 12 August, 2007.

Züllighoven, H. 2003. *Object-Oriented Construction Handbook: Developing Application-Oriented Software with the Tools & Materials Approach.* 1st edition. Morgan Kaufmann.

Zucconi L. 1995. Software Process Improvement Paradigms for IT Industry: Why the Bottom-Up Approach Fits Best. *Proceedings of the 1995 Asia Pacific Software Engineering Conference (APSEC '95): IEEE*, CSIRO Division of information Technology.

APPENDICES

Appendix – 1: Survey Questionnaires

Appendix 1A: Questionnaire for Software Development Professionals

Dear Madam/Sir

A research is currently being undertaken to understand, support and improve the software development practice/situation in Ethiopia.

The overall purposes of the research are,

- to better understand the existing software development practices locally,
- to learn about and critically assess the practical use, usability and appropriateness (affordability, availability, sustainability, etc.) of popular ('industry standard') methodologies, in the local settings, and
- to use the findings to propose and test (in real-life projects) context sensitive and usable innovative methodical approaches that would help in improving the software development situation in our country.

To help in this effort, the enclosed questionnaire is being circulated to organizations and professionals that are actively involved in the development of software locally. This survey questionnaire is just one of the instruments being used to provide insight and will soon be followed by interviews and workshops with major actors and analysis/diagnosis of selected real-life projects.

As one of the major actors in this area, you are kindly requested to participate in this survey. All you need to do for this moment is complete this questionnaire, which should not take you more than 30 minutes. Your responses will be kept confidential.

It would be greatly appreciated if you could complete the questionnaire by 23rd May 2007.

Should you have any questions or concerns in completing the questionnaire, please call 0911211327 or email to tesfayeb@ethionet.et

Thank you in advance for your time and effort in completing this survey questionnaire.

Sincerely,

Tesfaye Biru

I. Formal training and software development experiences

1. What is your highest education level?

☐ 1. High school

☐ 2. Two-year College

☐ 3. Four-year College

☐ 4. Master's Degree

☐ 5. Doctorate/PhD Degree

2. If you have a degree, what is your field of study?

☐ 1. Computer Science

☐ 2. Information Systems/Science

☐ 3. Computer Engineering

☐ 4. Management Information Systems

☐ 5. Other, please specify ______________________________

3. What is your year of experience in systems and software development

☐ 1. Less than two years ☐ 2. Between 2 and 5 years

☐ 3. More than five years

4. In how many projects have you involved so far? ___________________

☐ 1. One project ☐ 2. Two projects

☐ 3. Three projects ☐ 4. More than three projects

5. How do you upgrade your skills in software development techniques and tools?

☐ 5.1 Formal training in higher education

☐ 5.2 Short term qualification training

☐ 5.3 Attending conferences and workshops

☐ 5.4 Personal effort (through reading books, tutorials, etc.)

☐ 5.5 Other, please specify, ______________________

6. List the training you have taken so far other than your formal education (use additional pages if necessary)

	Type/area of training	Duration	Place of training
6.1			
6.2			
6.3			
6.4			
6.5			

7. How do you find the training provided to carry out your job?

☐ 1. Adequate

☐ 2. Inadequate

8. Does your company/institute pay for conferences and training

☐ 8.1 For Scientific conferences

☐ 8.2 For Computer/trade conferences

☐ 8.3 For Computer software/hardware vendor training on/off-site

☐ 8.4 For Short term professional training

☐ 8.5 The Company does not pay, I pay

II. Software development Projects

9. Please list the software projects you are involved in so far

No.	Project name	Specific role (eg. Programmer, analyst, designer, project manager, etc.	Project Duration	
			Start date	Finish date
9.1				
9.2				
9.3				
9.4				

10. For the projects you are involved in, have you participated in the planning stages?

☐ 1. In most of them

☐ 2. In some of them

☐ 3. In none of them

11. How do you report project progress?

☐ 11.1 Submit regular reports per plan

☐ 11.2 Weekly briefings

☐ 11.3 Do not report

☐ 11.4 Other, please specify

12. Which of the following software processes do you usually follow in the projects?

☐ 12.1. Simple code and-fix

☐ 12.2 Waterfall

☐ 12.3 Incremental and iterative

☐ 12.4 Unified process

☐ 12.5 Non-standard (eg. simple in house guideline)

☐ 12.6 Other, please specify ___________________________

13. Which of the following software methodologies do you usually use in the projects?

☐ 13.1. Structured systems analysis and design

☐ 13.2 Object oriented analysis and design

☐ 13.3 Select and combine several methods as required

☐ 13.4 Non-standard (eg. simple in house guideline)

☐ 13.5 Other, please specify: ___________________________

14. Which of the following software development tools do you usually use?

- ☐ 14.1 UML
- ☐ 14.2 Rational Rose
- ☐ 14.3 CASE
- ☐ 14.4 Microsoft project management
- ☐ 14.5 Other, please specify ______________________________

15. Which one of the following is your primary development language in the projects?

- ☐ 15.1 Java
- ☐ 15.2 C-Sharp
- ☐ 15.3 Visual Basic
- ☐ 15.4 Database development tools
- ☐ 15.5 Other, please specify _______________________

16. If you are not using standard methods, techniques and tools, the reason is

- ☐ 16.1 Lack of training
- ☐ 16.2 The company does not require
- ☐ 16.3 Tools are not available
- ☐ 16.4 It is time consuming (require more effort to learn and apply)
- ☐ 16.5 Not suitable – involve unnecessary steps/details and lack essential elements for local settings
- ☐ 16.6 Other, please specify ______________________________

17. What tools do you usually use to document and track project performance/schedule?

- ☐ 17.1 Gannt charts
- ☐ 17.2 PERT
- ☐ 17.3 Review meetings

18. Do you finish your specific project assignments on time?

☐ 18.1 Always

☐ 18.2 Most of the time

☐ 18.3 Some times

☐ 18.4 Not at all

19. If you do not mostly finish your project assignments on time, what are the reasons?

☐ 19.1 Difficulty in getting users to express their needs timely/clearly

☐ 19.2 Difficulty in capturing workarounds (work practices created by users to handle exceptional cases but not properly documented)

☐ 19.3 Frequently changing requirements

☐ 19.4 Lack of required diversity in skills within the project team

☐ 19.5 Unrealistic original schedule or limited resource allocation

☐ 19.6 High staff turnover

☐ 19.7 High work overload

☐ 19.8 Conflicts/uncertainties that result in group communication breakdown

☐ 19.9 Lack of properly defined roles and responsibilities

☐ 19.10 Inadequate communication and interaction with users

☐ 19.11 Low motivation among user professionals to participate

☐ 19.12 Weak managerial/organizational support

III. Communication with Users

20. How do you share information, experiences and learning with users?

- ☐ 20.1 Regular Meeting
- ☐ 20.2 Workshops
- ☐ 20.3 Planned discussions
- ☐ 20.4 Informal meetings and discussions
- ☐ 20.5 Other, pleas specify, ___________________________

21. How often do you meet and work with users?

☐ 1. On daily basis	☐ 2. On weekly basis
☐ 3. On fortnightly basis	☐ 4. On monthly basis
☐ 5. Whenever necessary	☐ 6. Other, please specify _____

22. Where do you think users contribute more?

- ☐ 22.1 Requirement gathering stage
- ☐ 22.2 Design stage
- ☐ 22.3 Implementation stage
- ☐ 22.4 Through out the development process
- ☐ 22.5 Other, please specify, ________________________________

23. How do you handle conflicts with users whenever they arise?

- ☐ 23.1 Confront and resolve
- ☐ 23.2 Withdraw from engagement
- ☐ 23.3 Report to management
- ☐ 23.4 Wait for some time until conflict subsides
- ☐ 23.5 Other, please specify

24. At which location do you find yourself to be more productive?

☐ 1. Working at the users' location

☐ 2. Working at your own location (away from the user's location)

25. If you are productive at the users' location, what do you think are the reasons?

☐ 25.1 Better interaction and collaboration with users

☐ 25.2 Better access to tools and facilities

☐ 25.3 Less interruption by other activities (more focused)

☐ 25.4 Other, please specify, ________________________________

IV. Communication within the Development Team

26. How often do you meet with your group members?

☐ 1. On a daily basis	☐ 2. On a weekly basis
☐ 3. On a fortnightly basis	☐ 4. On a monthly basis
☐ 5. Whenever necessary	☐ 6. Other, please specify _____

27. How do you share information, experiences and learning with group members?

☐ 27.1 Regular Meeting

☐ 27.2 Workshops

☐ 27.3 Planned discussions

☐ 27.4 Informal meetings and discussions

☐ 27.5 Use of modern technology (groupware, email, etc.)

☐ 27.6 Other, please specify, _________________________

28. During team meetings, members

☐ 28.1 Express thoughts and opinions clearly and freely

☐ 28.2 Are shy and reserved

☐ 28.3 Are influenced by groupthink (avoid critical thinking, promote consensus thinking, etc.)

☐ 28.4 Other, please specify ____________________

V. General

29. Which of the following guidelines are available and in use at your workplace?

☐ 29.1 Planning guideline

☐ 29.2 Change management guidelines

☐ 29.3 Requirement documentation guidelines

☐ 29.4 Design documentation guideline

☐ 29.5 Coding guideline

☐ 29.6 No guideline

30. If you have guidelines, how do you insure proper use?

☐ 30.1 Technical walkthrough

☐ 30.2 Peer review (rotate specs among peers for comment/review)

☐ 30.3 Management review

☐ 30.4 User feedback

☐ 30.5 Other, please specify, ________________________________

31. From your experience, which aspects of software development are challenging?

	Very challenging	Challenging	Not challenging
31.1 Planning			
31.2 Requirement gathering			
31.3 Analysis and design			
31.4 Coordination with the team			
31.5 Communication with users			
31.6 Use of tools			
31.7 Coding			
31.8 Other, please specify			

32. Which of the following do you think are the most critical limiting factors to carry out your task effectively and efficiently?

- ☐ 32.1 Absence of guidelines on methodology use
- ☐ 32.2 Lack of proper training
- ☐ 32.3 Lack of proper communication and collaboration
- ☐ 32.4 Lack of proper leadership and management
- ☐ 32.5 inappropriate personal relations and emotions
- ☐ 32.6 Lack of documentation skill
- ☐ 32.7 Lack of properly defined roles and responsibilities
- ☐ 32.8 Low level of trust between developers and users
- ☐ 32.9 Difficulty to cope up with the fast pace of technology change
- ☐ 32.10 Low motivation among user professionals to participate
- ☐ 32.11 Other, please specify, ______________________

33. In your opinion, which of the following areas need urgent intervention to improve the situation?

- ☐ 33.1 Adoption of guidelines and standards
- ☐ 33.2 Discipline approach towards project management
- ☐ 33.3 Skill upgrading / training
- ☐ 33.4 Provision of software tools
- ☐ 33.5 Establishment of mutual understanding with users
- ☐ 33.6 Other, please specify

34. In your opinion, what should be done to increase the usability of standard methodologies?

- ☐ 34.1 Provide tailored training on specific methodologies
- ☐ 34.2 Customize methodologies to local settings and requirements

☐ 34.3 Developing frameworks and guidelines to select and adapt methodologies to specific project settings

☐ 34.4 Develop and enforce national and organizational standards and guidelines for software processes and methods

☐ 34.5 Increase the availability and accessibility of tools

☐ 34.6 Create forums/platforms to exchange information, experience and learning among practitioners, trainers and researchers

☐ 34.7 Other, please specify, ________________

35. From your experience, what skills are important for software development?

	Very important	Important	Less Important
35.1 Knowledge of the application domain (the business area)			
35.2 Knowledge of development techniques and tools			
35.3 Business process improvement / reengineering skills			
35.4 Project management skills			
35.5 Analysis and design skills			
35.6 Programming skills			
35.7 Communication and negotiation skills			
35.8 Relationship building skills			
35.9 Other, please specify			

THE END, THANK YOU!

Appendix 1B: Questionnaire for Software Companies[19]

Dear Madam/Sir

A research is currently being undertaken to understand, support and improve the software development practice/situation in Ethiopia.

The overall purposes of the research are,

- to better understand the existing software development practices locally,
- to learn about and critically assess the practical use, usability and appropriateness (affordability, availability, sustainability, etc.) of popular ('industry standard') methodologies, in the local settings, and
- to use the findings to propose and test (in real-life projects) context sensitive and usable innovative methodical approaches that would help in improving the software development situation in our country.

To help in this effort, the enclosed questionnaire is being circulated to organizations and professionals that are actively involved in the development of software locally. This survey questionnaire is just one of the instruments being used to provide insight and will soon be followed by interviews and workshops with major actors and analysis/diagnosis of selected real-life projects.

As one of the major actors in this area, you are kindly requested to participate in this survey. All you need to do for this moment is complete this questionnaire, which should not take you more than 30 minutes. Your responses will be kept confidential.

It would be greatly appreciated if you could complete the questionnaire by 23rd May 2007.

Should you have any questions or concerns in completing the questionnaire, please call 0911211327 or email to tesfayeb@ethionet.et

Thank you in advance for your time and effort in completing this survey questionnaire.

Sincerely,

Tesfaye Biru

[19] A slightly modified version of this questionnaire is distributed to IT Departments of Government Organizations

I. Company Information

1. Name of the company: ____
2. Date of Establishment: ____
3. Address: ____

   ____
4. Number of people in the software development group

4.1 By Sex	Sex	No. of Staff
	1 Male	
	2 Female	
4.2 By type of employment	Type of employment	No. of Staff
	1 Full time	
	2 Part time	
4.3 By Educational Level	Educational level	No. of Staff
	1 High School	
	2 Diploma (two-year college)	
	3 B.Sc/BA	
	4 MSc./MA	
	5 PhD	
4.4 By Year of Graduation	Year	No. of staff
	1. 2006 and after	
	2. 2003-2005	
	3. 2000-2002	
	4. 1999 and before	
4.5 By age	Age	No. of staff
	1. 17-23	
	2. 24-30	
	3. 31-38	
	4. > 38	
4.6 By job category	Job category	No. of Staff
	4.6a Programmers	
	4.6b System Analysts	
	4.6c Project managers	
	4.6d General services and administrative support	

5. Number of **PROFESSIONAL** staff that left your company over the last five years

Year	Number of staff who left with out completing project assignments	Number of staff who left after completion of project assignments	Job categories most of the professionals who left (eg. Programmers, analysts, project managers)

6. List area of specialization (type of application softwares your company specialize in): ______________________________

7 Number of customers

Sector	No.	Category (please tick as appropriate)		
		Government	NGO	Private
7.1 Education				
7.2 Health				
7.3 Financial				
7.4 Civil Service				
7.5 Transport				
7.6 Publishing industry				
7.7 Factory				
7.8 Other, please specify				

8. Do you have partnership arrangement with other companies?

☐ 8.1 Yes, with foreign software development companies

☐ 8.2 Yes, with local software companies

☐ 8.3 Yes, Other, please specify ________________________

☐ 8.4 No partnership arrangement

9. Means of getting new customer:

☐ 9.1 Through participation in competitive tendering process

☐ 9.2 Through personal contact

10. If your response to question number 9 is through tendering process, what is your opinion about the tendering process?

- ☐ 10.1 Tender documents provide adequate information on user requirements to prepare responsive proposals.
- ☐ 10.2 Tender documents usually demand use of standard methodologies and processes
- ☐ 10.3 Tender processing usually takes very long time (beyond the validity dates of the proposals)
- ☐ 10.4 Tender documents usually provide enough time for preparing proposals
- ☐ 10.5 Tender documents specify realistic project duration
- ☐ 10.5 Tender documents clearly specify general contract terms and conditions
- ☐ 10.6 Tender documents contain flexible contract terms and conditions
- ☐ 10.7 Other, please specify: ________________________________

11. For which of the following activities do you have formally documented policy and procedure?

- ☐ 11.1 Project initiation, planning, monitoring and control
- ☐ 11.2 Requirement gathering
- ☐ 11.3 Analysis and design
- ☐ 11.4 Coding
- ☐ 11.5 Testing and handover
- ☐ 11.6 Maintenance and support
- ☐ 11.7 No formally documented policy and procedure

12. If you have formally documented policies and procedures, please indicate whether you have proper guidelines and training for their use

☐ 12.1 Appropriate guidelines for use are available and accessible

☐ 12.2 Guidelines are updated regularly

☐ 12.3 Training on use of guidelines are provided regularly

☐ 12.4 Orientation is provided for project personnel upon assignment

☐ 12.5 There are no guidelines for use

13. Do you prepare project management plan for your software development projects?

☐ 1. Always

☐ 2. Not always, only when requested by the users/client

☐ 3. No separate project management plan (We simply use the scope of work in the contract)

14. In relation to assignment of project management responsibility?

☐ 14.1 A project manager is assigned for each project

☐ 14.2 Software unit manager plays the role of the project manager

15. What are the deliverables usually expected of projects (in addition to the software)?

☐ 15.1 Inception report

☐ 15.2 Project management plan

☐ 15.3 Requirement analysis document

☐ 15.4 Design document

☐ 15.5 Test plans and results

☐ 15.6 Project progress report

☐ 15.7 Operation and training manual

☐ 15.8 Other, please specify ______________________________

16. How do you handle changes in projects

☐ 16.1 There is a standard format to initiate, discuss and approve change

☐ 16.2. The change process is communicated to all stakeholders timely

☐ 16.3 Only management participates in the decision process related to change

☐ 16.4 Both management and users participate in the decision process related to change

☐ 16.5 The development team also participates in the decision process

☐ 16.6 On the basis of approved changes, the project plan and contract are formally revised and properly documented

☐ 16.7 Additional resources required are provided immediately

17. In relation projects already completed (please fill in the following table for upto five most current projects)

Project name	Planned		Actual		Major Reasons for delay (please tick the reasons that apply – see the description at the bottom part of the page)
	Start	Finish	Start	Finish	
					☐A ☐B ☐C ☐D ☐E ☐F ☐G ☐H
					☐A ☐B ☐C ☐D ☐E ☐F ☐G ☐H
					☐A ☐B ☐C ☐D ☐E ☐F ☐G ☐H

Major reasons for project delay

A. Poor planning because of unclear / incomplete requirements

B. Poor planning because of lack of project management skill

C. Lack of proper progress monitoring and control
D. High project staff turnover
E. Lack of use of appropriate software development methodology
F. Inadequate budget to finance additional work due to changes in user requirements
G. Lack of cooperation from users
H. Problems within the development team

18. For completed projects, how do you compare the alterations/changes in the requirement specified in the original tender document and actual implementation?

☐ 18.1 Almost the same

☐ 18.2 Similar but with minor variation

☐ 18.3 Major variation

☐ 18.4 Totally different

19. If your response to question 18 is "major variation" or "totally different", what do you think are the major reasons?

☐ 19.1 Inability of the user to adequately articulate requirements at the beginning

☐ 19.2 Lack of requirement engineering skills of developers

☐ 19.3 Inadequate planning during the tendering

☐ 19.4 Changes in the organization since the commencement of the project

☐ 19.4 Other, please specify: ______________________________

20. Please indicate the relative importance of the following factors in terms of helping you deliver quality product on time and within budget

		Very important	important	Not important
20.1	Introduce and follow standard / formal methods for software development			
20.2	Introduce and follow disciplined software development project management			
20.3	Introduce skill upgrading programs for software staff in technical areas.			
20.4	Introduce skill upgrading programs for software staff in organization and management			
20.5	Partner with experienced foreign software development company			
20.6	Work closely with higher learning institutions to improve the profiles of graduates			
20.7	Increase the use of software development technologies and tools			
20.8	Properly define and institutionalize the software development process			
20.9	Involve user representatives in the development team			
20.10				

Appendix – 2: Sample Source Programs

```
/************************************
*       windio.c - Low level input output routines
************************************/
#include <c:\qc25\include\stdio.h>
#include <c:\qc25\include\dos.h>
#include <c:\qc25\include\string.h>
#include "windows.h"

static void initcur(void);

static int cursorstart = -1, cursorend = -1;

void cursoroff()
{
        union REGS regs;
        initcur();
        regs.h.ah = 1;
        regs.x.cx = 0x2000;
        int86(0x10, &regs, &regs);
}
void cursoron()
{
        union REGS regs;

        initcur();
        regs.h.ah = 1;
        regs.h.ch = cursorstart;
        regs.h.cl = cursorend;
        int86(0x10, &regs, &regs);
}
void setcurpos(row, col)
int row;
int col;
{
        union REGS regs;

        regs.h.ah = 2;
        regs.h.bh = 0;
        regs.h.dh = --row;
        regs.h.dl = --col;
        int86(0x10, &regs, &regs);
}

void setcursor(cstart, cend)
```

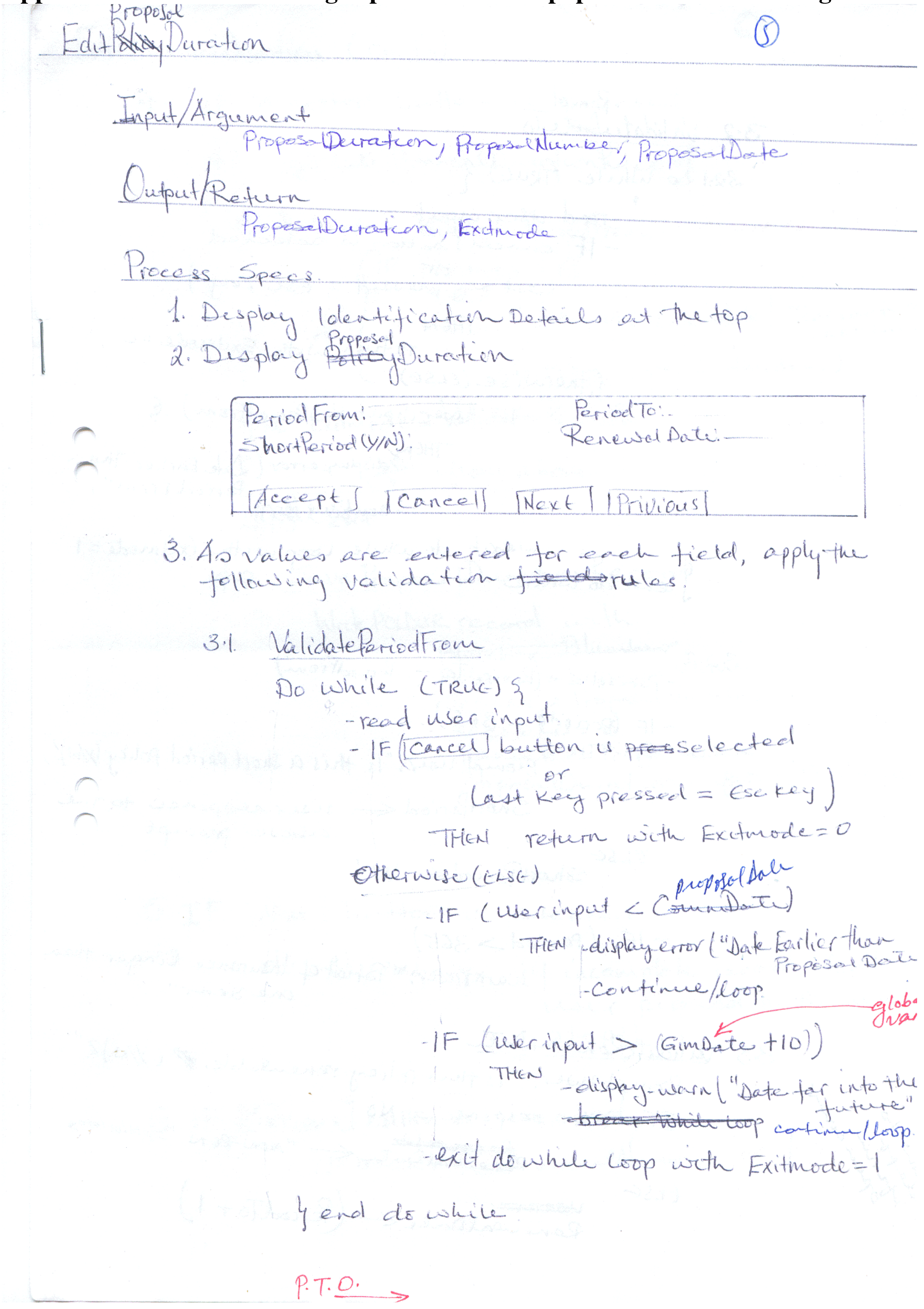
```
int cstart;
int cend;
{
        cursorstart = cstart;
        cursorend = cend;
        cursoron();
}
void getcurpos(row, col, cstart, cend)
int *row;
int *col;
int *cstart;
int *cend;
{
        union REGS regs;
        regs.h.ah = 3;
        regs.h.bh = 0;
        int86(0x10, &regs, &regs);
        *row = ++regs.h.dh;
        *col = ++regs.h.dl;
        *cstart = regs.h.ch;
        *cend = regs.h.cl;
}

void fillone(row, col, chr, att)
int row;
int col;
int chr;
int att;
{
        union REGS regs;
        setcurpos(row, col);
        regs.h.ah = 9;
        regs.h.al = chr;
        regs.h.bh = 0;
        regs.h.bl = att;
        regs.x.cx = 1;
        int86(0x10, &regs, &regs);
}
void printone(row, col, chr)
int row;
int col;
int chr;
{
        union REGS regs;

        setcurpos(row, col);
        regs.h.ah = 10;
        regs.h.al = chr;
        regs.h.bh = 0;
        regs.x.cx = 1;
        int86(0x10, &regs, &regs);
}
void setone(row, col, att)
int row;
int col;
int att;
{
        union REGS regs;
        setcurpos(row, col);
        regs.h.ah = 8;
        regs.h.bh = 0;
        int86(0x10, &regs,  &regs);
        regs.h.ah = 9;
        regs.h.bl = att;
        regs.x.cx = 1;
        int86(0x10, &regs, &regs);
}
void printcenter(row, col, string)
int row;
int col;
char *string;
{
        printstring(row, col-(strlen(string) >> 1), string);
}
static void initcur()
{
        union REGS regs;
        if (cursorstart == -1 && cursorend == -1) {
                regs.h.ah = 3;
                regs.h.bh = 0;
                int86(0x10, &regs, &regs);
                cursorstart = regs.h.ch;
                cursorend = regs.h.cl;
        }
}
```

```
/********************************
*	window.c - Dynamic window routines
*********************************/
#include <c:\qc25\include\stdio.h>
#include <c:\qc25\include\stdlib.h>
#include <c:\qc25\include\stdarg.h>
#include <c:\qc25\include\malloc.h>
#include "windows.h"
static void reset_initial_video(void);
void draw_window(row1, col1, row2, col2, watt, bflg, ...)
int row1, col1;
int row2, col2;
int watt;
int bflg;
{
	int batt;
	va_list arg_marker;
	va_start(arg_marker, bflg);
	clearscreen(row1, col1, row2, col2, watt);
	if (bflg != _NO_BORDER) {
		batt = va_arg(arg_marker, int);
		drawbox(row1, col1, row2, col2, bflg, batt);
	}
}
void draw_window(int, int, int, int, int, int, ...);
WINDOW *open_window(row1, col1, row2, col2, draw, ...)
int row1, col1;
int row2, col2;
int draw;
{
	int watt, bflg, batt;
	va_list arg_marker;
	WINDOW *window;
	va_start(arg_marker, draw);
	window = malloc(sizeof(WINDOW));
	if (window == NULL) {
		display_error("Not enough memory to open window");
		return(window);
	}
	window->row1 = row1;
	window->col1 = col1;
	window->row2 = row2;
	window->col2 = col2;
	window->videoarray = malloc((col2 - col1 + 1) * 2 * (row2 - row1 + 1));
	if (window->videoarray == NULL) {
		free(window);
```

```
                display_error("Not enough memory to open window");
                return(window);
        }
        savescreen(row1, col1, row2, col2, window->videoarray);
        if (draw) {
                watt = va_arg(arg_marker, int);
                bflg = va_arg(arg_marker, int);
                if (bflg == _NO_BORDER)
                        draw_window(row1, col1, row2, col2, watt, _NO_BORDER);
                else  {
                        batt = va_arg(arg_marker, int);
                        draw_window(row1, col1, row2, col2, watt, bflg, batt);
                }
        }
        return(window);
}
WINDOW *open_window(int, int, int, int, int, ...);
WINDOW  *close_window(window)
WINDOW *window;
{
        if (window != NULL) {
                restorescreen(window->row1, window->col1, window->row2,
                        window->col2, window->videoarray);
                free(window->videoarray);
                free(window);
        }
        return(NULL);
}
#ifdef MICROSOFTC
#define DEFMEMMOVE
#endif
#ifdef DEFMEMMOVE
        static char *memmove(dst, src, n)
        char *dst;
        char *src;
        unsigned int n;
        {
                char *beg = src;
                if (src + n > dst) {
                        src +=n;
                        dst +=n;
                        while (n--)
                                *--dst = *--src;
                }
                else
                        while (n--)
                        *dst++ = *src++;
                return(beg);
        }
#endif
void scroll_window(window, num, dir, att)
WINDOW *window;
int num;
int dir;
int att;
{
        int i, row1, col1, row2, col2, rows, cols;
        char *videoarray;
        switch (dir) {
                case _UP:
                case _DOWN:
                case _LEFT:
                case _RIGHT:
                        row1 = window->row1 + 1;
                        col1 = window->col1 + 1;
                        row2 = window->row2 - 1;
                        col2 = window->col2 - 1;
                        break;
                case _UPA:
                case _DOWNA:
                case _LEFTA:
                case _RIGHTA:
                        row1 = window->row1;
                        col1 = window->col1;
                        row2 = window->row2;
                        col2 = window->col2;
        }
        cols = (col2 - col1 + 1) * 2;
        rows = row2 - row1 + 1;
        if ((videoarray = malloc(cols * rows)) == NULL) {
                display_error("Not enough memory to allocate scroll buffer");
                return(window);
        }
        savescreen(row1, col1, row2, col2, videoarray);
        switch (dir) {
                case _UP:
                case _UPA:
                        for (i = row1 + num; i < row2 + 1; i++)
                                memmove(videoarray + (i - num - row1) * cols,
                                        videoarray + (i - row1) * cols, cols);
                        break;
                case _DOWN:
```

```
		case _DOWNA:
			for (i = row2; i >= row1 + num; i--)
				memmove(videoarray + (i - row1) * cols,
					videoarray + (i - num - row1) * cols, cols);
			break;
		case _LEFT:
		case _LEFTA:
			for (i = row1; i <= row2; i++)
				memmove(videoarray + (i - row1) * cols,
					videoarray + (i - row1) * cols + num * 2,
					cols - num * 2);
			break;
		default:
			for (i = row1; i <= row2; i++)
				memmove(videoarray + (i - row1) * cols + num * 2,
					videoarray + (i - row1) * cols, cols - num * 2);
	}
	restorescreen(row1, col1, row2, col2, videoarray);
	if (att) {
		switch (dir) {
			case _UP:
			case _UPA:
				clearscreen(row2 - num + 1, col1, row2, col2, att);
				break;
			case _DOWN:
			case _DOWNA:
				clearscreen(row1, col1, row1 + num -1, col2, att);
				break;
			case _LEFT:
			case _LEFTA:
				clearscreen(row1, col2 - num + 1, row2, col2, att);
				break;
			default:
				clearscreen(row1, col1, row2, col1 + num - 1, att);
		}
	}
	free(videoarray);
}
void vertical_bar(window, current, total, att)
WINDOW *window;
int current;
int total;
int att;
{
	int marker;
	if (total == 0) {
		current = 0;
		total = 1;
	}
	fillone(window->row1 + 1, window->col2, 24, att);
	fillscreen(window->row1 + 2, window->col2, window->row2 - 1,
		window->col2, 177, att);
	fillone(window->row2 - 1, window->col2, 25, att);
	marker = (int)((long)(window->row2 - window->row1 - 4)
		* current / total + window->row1 + 2);
	fillone(marker, window->col2, 176, att);
}
void horizontal_bar(window, current, total, att)
WINDOW *window;
int current;
int total;
int att;
{
	int marker;
	if (total == 0) {
		current = 0;
		total = 1;
	}
	fillone(window->row2, window->col1 + 1, 27, att);
	fillscreen(window->row2, window->col1 + 2, window->row2,
		window->col2 - 2, 177, att);
	fillone(window->row2, window->col2 - 1, 26, att);
	marker = (int)((long)(window->col2 - window->col1 - 4)
		* current / total + window->col1 + 2);
	fillone(window->row2, marker, 176, att);
}
static WINDOW *window;
static int srow, scol, sstart, send;
void save_initial_video()
{
	settext80();
	getcurpos(&srow, &scol, &sstart, &send);
	cursoroff();
	window = open_window(1, 1, 25, 80, _DRAW, 7, _NO_BORDER);
	atexit(reset_initial_video);
}
static void reset_initial_video()
{
	close_window(window);
	setcurpos(srow, scol);
	setcursor(sstart, send);
}
```

Appendix – 3: Insurance design specifications on paper and Email exchanges



Edit ~~Policy~~ Proposal Duration (5)

Input/Argument
ProposalDuration, ProposalNumber, ProposalDate

Output/Return
ProposalDuration, Exitmode

Process Specs.

1. Display Identification Details at the top
2. Display ~~Policy~~ Proposal Duration

Period From:	Period To:
ShortPeriod (Y/N):	Renewal Date:
[Accept] [Cancel]	[Next] [Privious]

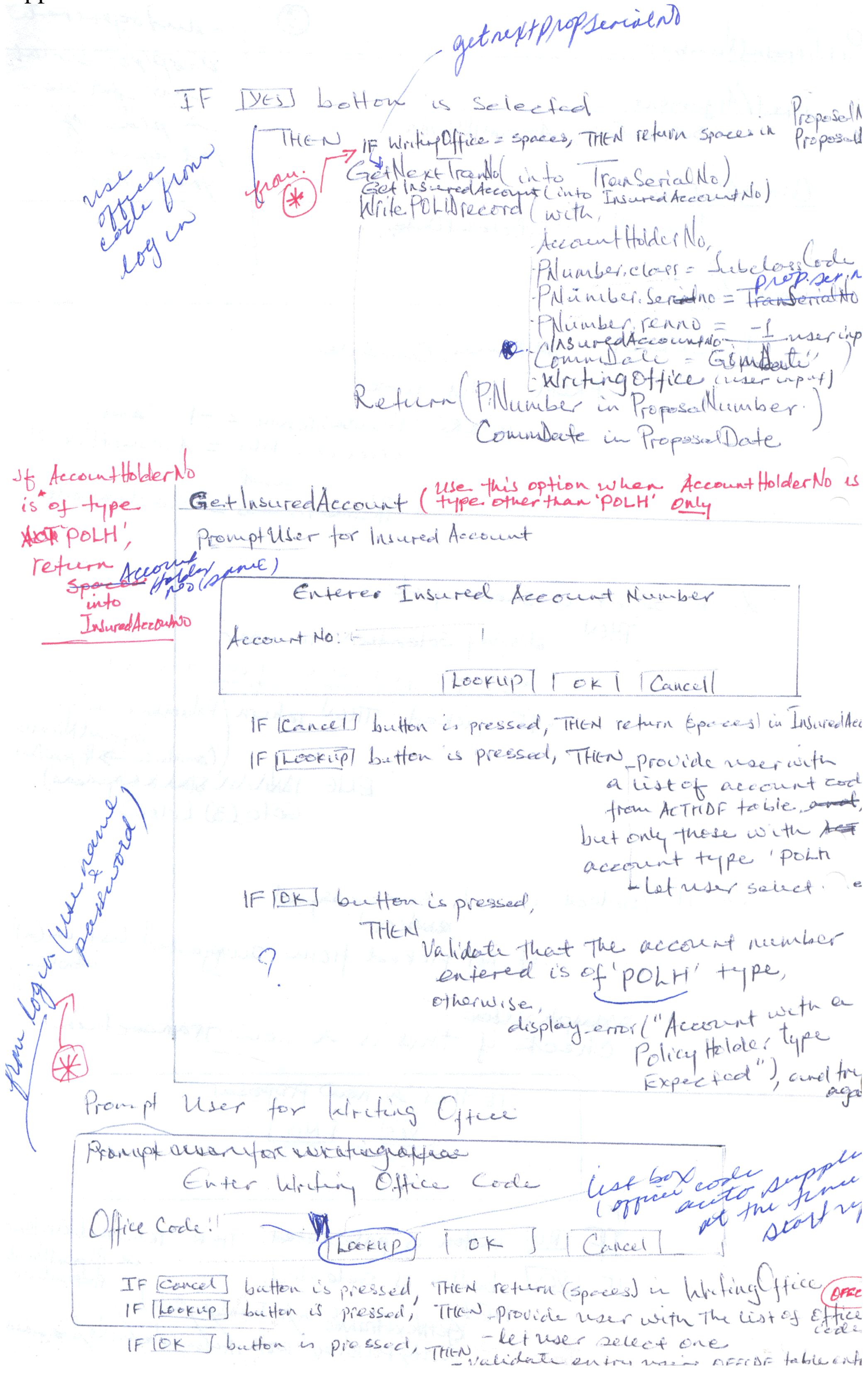
3. As values are entered for each field, apply the following validation ~~fields~~ rules.

3.1 ValidatePeriodFrom

```
Do while (TRUE) {
    - read user input
    - IF ([Cancel] button is ~~press~~ selected
              or
         Last Key pressed = Esc key)
        THEN return with Exitmode = 0
  Otherwise (ELSE)
    - IF (User input < ~~CommDate~~ proposalDate)
        THEN - display error ("Date Earlier than Proposal Date
             - Continue/loop
    - IF (User input > (GimDate + 10))
        THEN - display-warn ("Date far into the future"
             ~~break while loop~~ continue/loop
    - exit do while Loop with Exitmode = 1
} end do while
```

globa var

P.T.O. →



getnextpropserialno

use office code from log in

```
IF [YES] button is selected
    THEN  IF WritingOffice = spaces, THEN return spaces in ProposalN
                                                          Proposal
          from: (*)
          GetNextTranNo( into TranSerialNo)
          Get InsuredAccount (into InsuredAccountNo)
          Write POLIDrecord ( with,
                  AccountHolderNo,
                  PNumber.class = SubclassCode
                  PNumber.SerialNo = ~~TranSerialNo~~ prop.ser.n
                  PNumber.renno = -1
                  InsuredAccountNo user inp
                  CommDate = GtmDate'
                  WritingOffice (user input)
          Return( PNumber in ProposalNumber.
                  CommDate in ProposalDate )
```

If AccountHolderNo is of type ~~ACT~~ 'POLH', return ~~Spaces~~ AccountHolderNo (same) into InsuredAccountNo

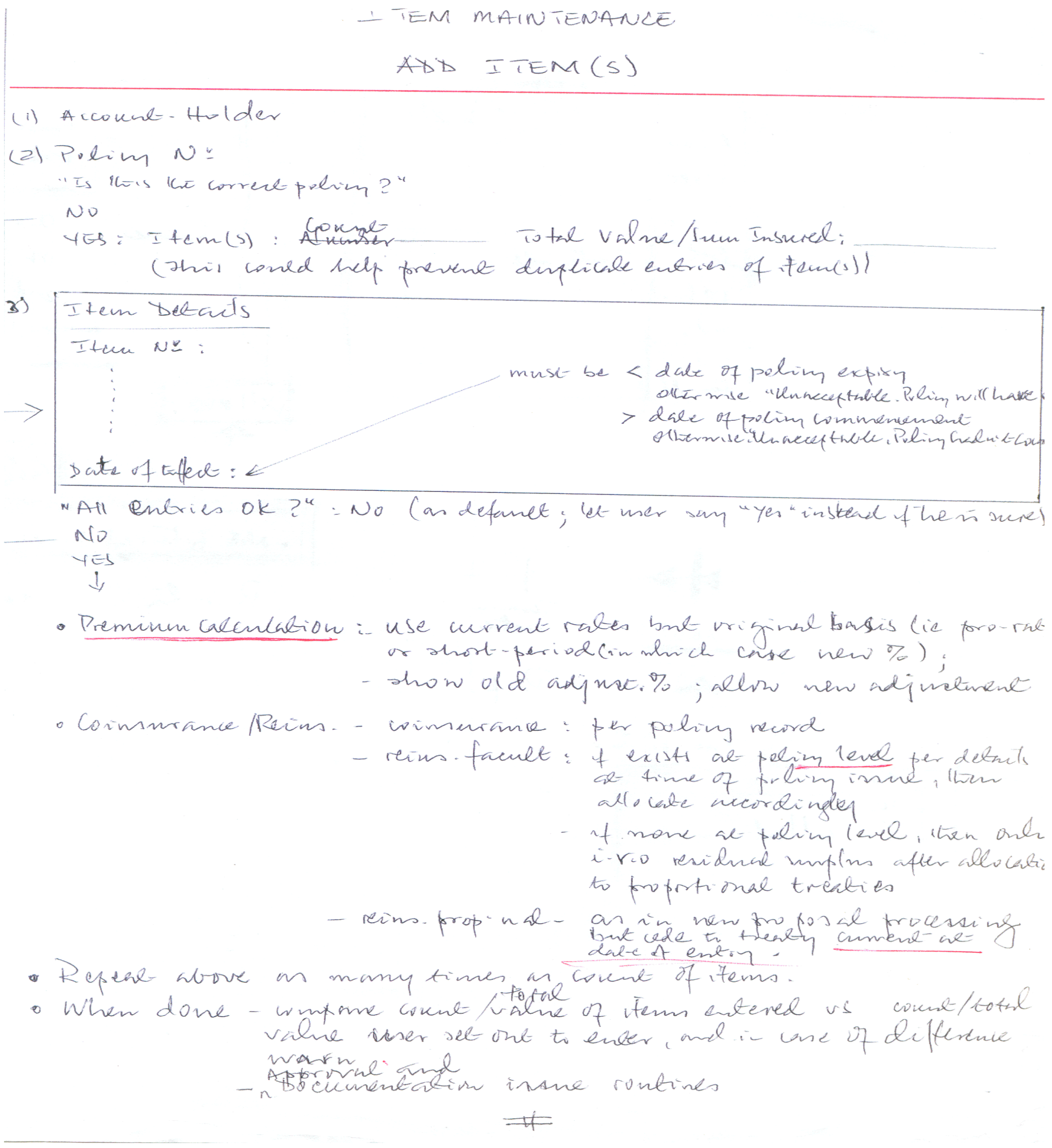
```
GetInsuredAccount ( Use this option when AccountHolderNo is type other than 'POLH' only
    PromptUser for Insured Account

    +-------------------------------------------+
    |        Enterer Insured Account Number     |
    | Account No:                               |
    |               [Lookup]  [ OK ]  [Cancel]  |
    +-------------------------------------------+

    IF [Cancel] button is pressed, THEN return (spaces) in InsuredAcc
    IF [Lookup] button is pressed, THEN - provide user with
                                          a list of account cod
                                          from ACTHDF table, ~~and~~,
                                          but only those with ~~ACT~~
                                          account type 'POLH
                                        - Let user select.
    IF [OK] button is pressed,
         THEN
    ?        Validate that the account number
             entered is of 'POLH' type,
             otherwise,
                 display_error("Account with a
                         Policy Holder type
                         Expected"), and try
                                          aga
```

from login (use name & password) (*)

```
Prompt User for Writing Office

    +-------------------------------------------+
    | ~~Prompt user for writingoffice~~         |
    |       Enter Writing Office Code           |
    | Office Code:     [Lookup]  [ OK ]  [Cancel]|
    +-------------------------------------------+

    IF [Cancel] button is pressed, THEN return (spaces) in WritingOffice  OFFC
    IF [Lookup] button is pressed, THEN - provide user with the list of Office
                                                                      code
                                        - let user select one
    IF [OK ] button is pressed, THEN
                                        - validate entry using OFFCDF table entr
```

list box (office code) auto supplied at the time start up

Appendix 3 ... cont'd



ITEM MAINTENANCE

ADD ITEM(S)

(1) Account-Holder

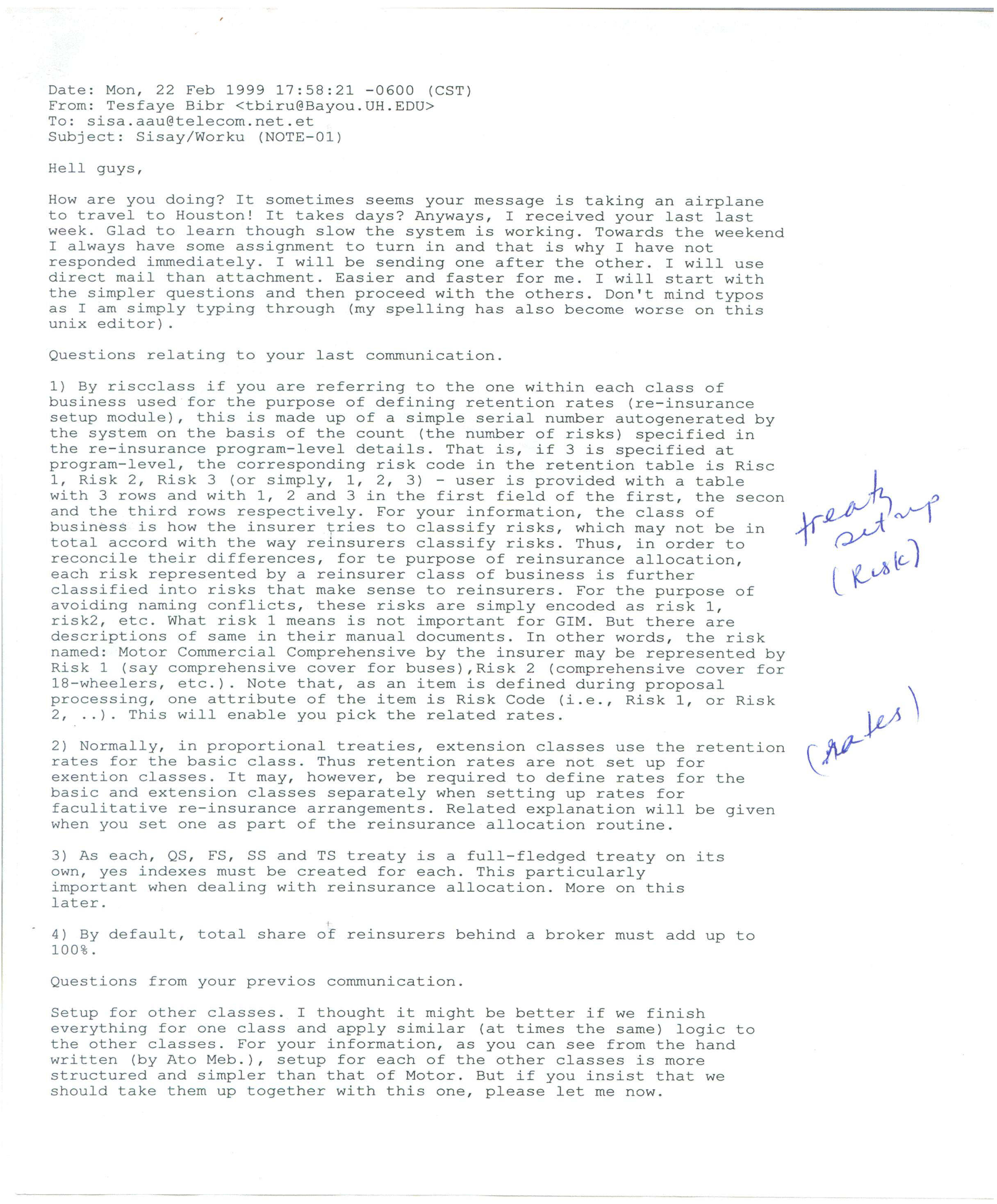
(2) Policy No:
"Is this the correct policy?"
NO
YES: Item(s): Count ~~Number~~ ______ Total Value/Sum Insured: ______
(This could help prevent duplicate entries of item(s))

3) Item Details
Item No:
:
:
Date of Effect: ← must be < date of policy expiry
otherwise "Unacceptable. Policy will have
> date of policy commencement
otherwise "Unacceptable, Policy hadn't com

"All Entries OK?" : No (as default; let user say "Yes" instead if he is sure)
NO
YES
↓

- Premium Calculation: use current rates but original basis (ie pro-rata or short-period (in which case new %);
  - show old adjust. %; allow new adjustment
- Coinsurance/Reins. - coinsurance: per policy record
  - reins. facult: if exists at policy level per details at time of policy issue, then allocate accordingly
    - if none at policy level, then only i.r.o residual surplus after allocation to proportional treaties
  - reins. prop'nal - as in new proposal processing but cede to treaty current at date of entry.
- Repeat above as many times as count of items.
- When done - compare count/total value of items entered vs count/total value user set out to enter, and in case of difference warn
  - Approval and Documentation issue routines

#



Date: Mon, 22 Feb 1999 17:58:21 -0600 (CST)
From: Tesfaye Bibr <tbiru@Bayou.UH.EDU>
To: sisa.aau@telecom.net.et
Subject: Sisay/Worku (NOTE-01)

Hell guys,

How are you doing? It sometimes seems your message is taking an airplane to travel to Houston! It takes days? Anyways, I received your last last week. Glad to learn though slow the system is working. Towards the weekend I always have some assignment to turn in and that is why I have not responded immediately. I will be sending one after the other. I will use direct mail than attachment. Easier and faster for me. I will start with the simpler questions and then proceed with the others. Don't mind typos as I am simply typing through (my spelling has also become worse on this unix editor).

Questions relating to your last communication.

1) By riscclass if you are referring to the one within each class of business used for the purpose of defining retention rates (re-insurance setup module), this is made up of a simple serial number autogenerated by the system on the basis of the count (the number of risks) specified in the re-insurance program-level details. That is, if 3 is specified at program-level, the corresponding risk code in the retention table is Risc 1, Risk 2, Risk 3 (or simply, 1, 2, 3) - user is provided with a table with 3 rows and with 1, 2 and 3 in the first field of the first, the secon and the third rows respectively. For your information, the class of business is how the insurer tries to classify risks, which may not be in total accord with the way reinsurers classify risks. Thus, in order to reconcile their differences, for te purpose of reinsurance allocation, each risk represented by a reinsurer class of business is further classified into risks that make sense to reinsurers. For the purpose of avoiding naming conflicts, these risks are simply encoded as risk 1, risk2, etc. What risk 1 means is not important for GIM. But there are descriptions of same in their manual documents. In other words, the risk named: Motor Commercial Comprehensive by the insurer may be represented by Risk 1 (say comprehensive cover for buses),Risk 2 (comprehensive cover for 18-wheelers, etc.). Note that, as an item is defined during proposal processing, one attribute of the item is Risk Code (i.e., Risk 1, or Risk 2, ..). This will enable you pick the related rates.

treaty setup (Risk)

2) Normally, in proportional treaties, extension classes use the retention rates for the basic class. Thus retention rates are not set up for exention classes. It may, however, be required to define rates for the basic and extension classes separately when setting up rates for faculitative re-insurance arrangements. Related explanation will be given when you set one as part of the reinsurance allocation routine.

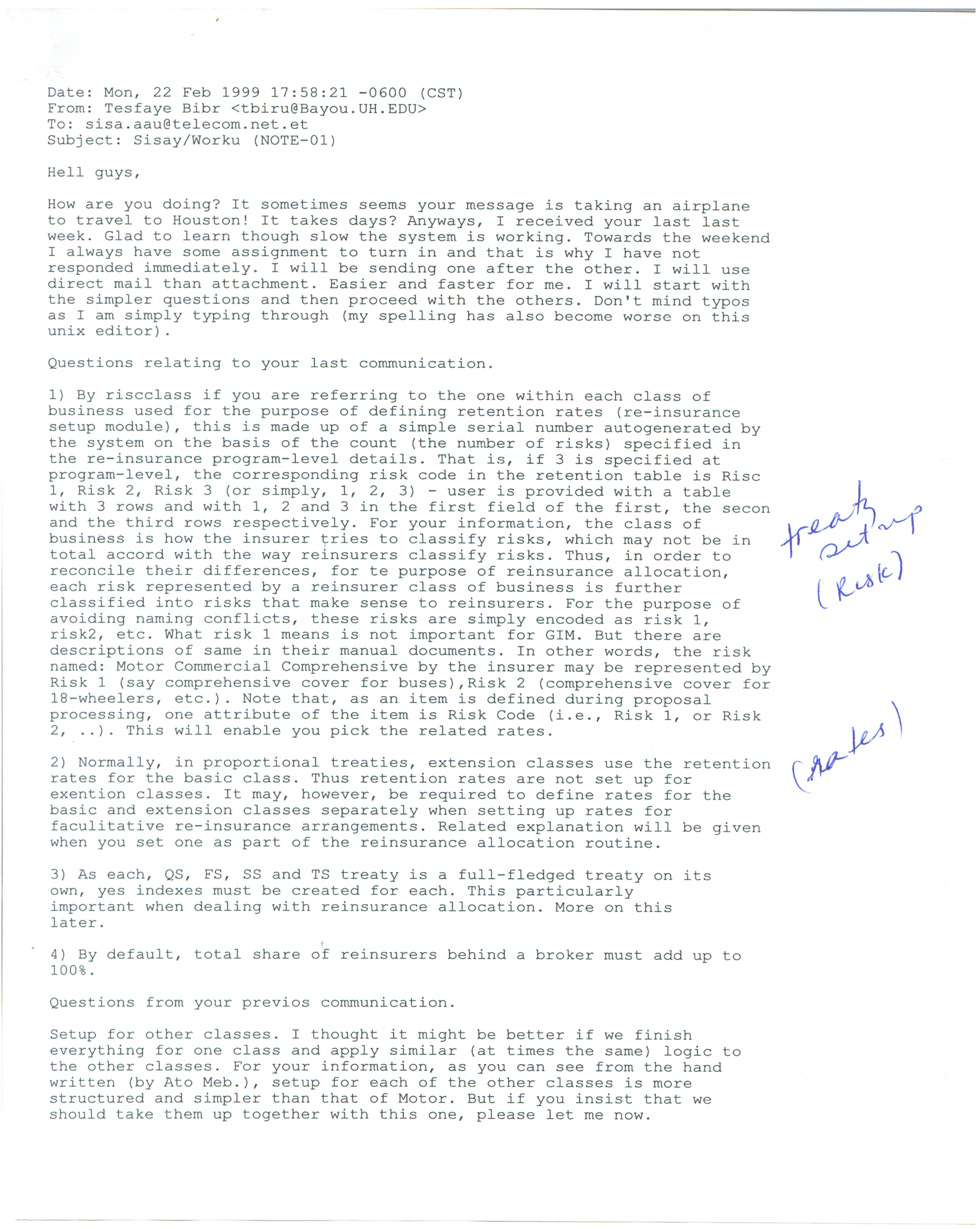
(rates)

3) As each, QS, FS, SS and TS treaty is a full-fledged treaty on its own, yes indexes must be created for each. This particularly important when dealing with reinsurance allocation. More on this later.

4) By default, total share of reinsurers behind a broker must add up to 100%.

Questions from your previos communication.

Setup for other classes. I thought it might be better if we finish everything for one class and apply similar (at times the same) logic to the other classes. For your information, as you can see from the hand written (by Ato Meb.), setup for each of the other classes is more structured and simpler than that of Motor. But if you insist that we should take them up together with this one, please let me now.



Date: Mon, 22 Feb 1999 17:58:21 -0600 (CST)
From: Tesfaye Bibr <tbiru@Bayou.UH.EDU>
To: sisa.aau@telecom.net.et
Subject: Sisay/Worku (NOTE-01)

Hell guys,

How are you doing? It sometimes seems your message is taking an airplane to travel to Houston! It takes days? Anyways, I received your last last week. Glad to learn though slow the system is working. Towards the weekend I always have some assignment to turn in and that is why I have not responded immediately. I will be sending one after the other. I will use direct mail than attachment. Easier and faster for me. I will start with the simpler questions and then proceed with the others. Don't mind typos as I am simply typing through (my spelling has also become worse on this unix editor).

Questions relating to your last communication.

1) By riscclass if you are referring to the one within each class of business used for the purpose of defining retention rates (re-insurance setup module), this is made up of a simple serial number autogenerated by the system on the basis of the count (the number of risks) specified in the re-insurance program-level details. That is, if 3 is specified at program-level, the corresponding risk code in the retention table is Risc 1, Risk 2, Risk 3 (or simply, 1, 2, 3) - user is provided with a table with 3 rows and with 1, 2 and 3 in the first field of the first, the secon and the third rows respectively. For your information, the class of business is how the insurer tries to classify risks, which may not be in total accord with the way reinsurers classify risks. Thus, in order to reconcile their differences, for te purpose of reinsurance allocation, each risk represented by a reinsurer class of business is further classified into risks that make sense to reinsurers. For the purpose of avoiding naming conflicts, these risks are simply encoded as risk 1, risk2, etc. What risk 1 means is not important for GIM. But there are descriptions of same in their manual documents. In other words, the risk named: Motor Commercial Comprehensive by the insurer may be represented by Risk 1 (say comprehensive cover for buses),Risk 2 (comprehensive cover for 18-wheelers, etc.). Note that, as an item is defined during proposal processing, one attribute of the item is Risk Code (i.e., Risk 1, or Risk 2, ..). This will enable you pick the related rates.

treaty setup (Risk)

2) Normally, in proportional treaties, extension classes use the retention rates for the basic class. Thus retention rates are not set up for exention classes. It may, however, be required to define rates for the basic and extension classes separately when setting up rates for faculitative re-insurance arrangements. Related explanation will be given when you set one as part of the reinsurance allocation routine.

(rates)

3) As each, QS, FS, SS and TS treaty is a full-fledged treaty on its own, yes indexes must be created for each. This particularly important when dealing with reinsurance allocation. More on this later.

4) By default, total share of reinsurers behind a broker must add up to 100%.

Questions from your previos communication.

Setup for other classes. I thought it might be better if we finish everything for one class and apply similar (at times the same) logic to the other classes. For your information, as you can see from the hand written (by Ato Meb.), setup for each of the other classes is more structured and simpler than that of Motor. But if you insist that we should take them up together with this one, please let me now.

Appendix - 4: Project Management Tools used for teaching and projects

Appendix 4A: Communications Plan

Communications Plan				
Senior Management				
Communication (Meeting, Report, etc.)	**Frequency / Dates**	**Originator**	**Distribution/ Information Flow**	**Comments**
Functional Management				
Communication (Meeting, Report, etc.)	**Frequency / Dates**	**Originator**	**Distribution/Information Flow**	**Comments**
Project Team				
Communication (Meeting, Report, etc.)	**Frequency / Dates**	**Originator**	**Distribution/Information Flow**	**Comments**

Appendix 4B: Change Control Form

Project Change Control Form		
Project Name:		
Project Number:		
Project Manager:		
Change Request:		
Description (background):		
Impact Assessment		
Impact on Service/Quality		
Impact on Schedule:		
Impact on Cost:		
Immediate Action Required? (Include communication/ notification of change requirements):		
Authorization		
Requested by:	***Name:*** ***Signature:***	***Date Requested:***
Approved by:	***Name:*** ***Signature:***	***Date Approved:***

Appendix 4C: Risk Management Plan

Risk Management Plan				
Risk Event	**Probability of Risk**	**Consequence of Risk**	**Recommended Action**	**Description of Action**
	Low, **M**oderate, **H**igh	**L**ow, **M**oderate, **H**igh	Accept, Avoid, Transfer, Reduce	
1.				
2.				
3.				
4.				
5.				

Appendix 4D: Issue Management Log

Issue Management Log

Prepared by:

Issue Numbe r	***Issue Description***	***Date Identified***	***Assigned To***	***Date Resolution Required***	***Resolutio n***	***Date Resolved***

Appendix 4E: Meeting/Workshop Evaluation Form

Meeting/Workshop Evaluation Form **Date:**_____________

1. Purpose of the Meeting	1	2	3	4	5	
Totally unclear as to the purpose of the objective						The purpose or objective of the meeting was well defined

2. Decided what we wanted to achieve	1	2	3	4	5	
By the end of the meeting, we still had no idea of what we wanted to achieve.						We decided what we wanted to achieve by the end of the meeting.

3. Meeting Preparation	1	2	3	4	5	
We were totally unprepared for this meeting						We were sufficiently prepared for this meeting

4. Meeting Effectiveness	1	2	3	4	5	
Disconnected information and Tangent discussions.						Crisp & focused presentation & discussion. Good flow of information.

5. Team Participation	1	2	3	4	5	
Little team participation in discussions.						Team participated actively in the meeting.

6. Ground Rules	1	2	3	4	5	
We violated many of our ground rules during the meeting.						It was evident that we were living by our ground rules.

7. Meeting Process	1	2	3	4	5	
Meeting started/finished late and/or incomplete attendance.						Meeting started & finished on time with consistent attendance.

8. Time Allocation	1	2	3	4	5	
Agenda item continually ran over allotted times.						Agenda items addressed per the allotted times.

9. Meeting Usefulness	1	2	3	4	5	
A complete waste of time.						The meeting was an effective use of my time.

10. Team Building	1	2	3	4	5	
The meeting was a chore.						We had fun.

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbst verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Hamburg, im Oktober 2008

Tesfaye Biru