Looking at the Broad Picture

Looking at the Broad Picture: Smarter Software Development for Irish Companies

By

Marty Sanders

Edited by

Ita Richardson and Mícheál Ó hAodha

CAMBRIDGE SCHOLARS

PUBLISHING

Looking at the Broad Picture:
Smarter Software Development for Irish Companies,
by Marty Sanders

This book first published 2011

Cambridge Scholars Publishing

12 Back Chapman Street, Newcastle upon Tyne, NE6 2XX, UK

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN (10): 1-4438-2397-X, ISBN (13): 978-1-4438-2397-5

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

The development and maintenance of computer software continue to be surrounded by mystery. Even simple software is still very complex. Dr. Chris Horn, co-founder of Iona Technologies and President of Engineers Ireland, described it this way:

> The English translation of Tolstoy's *War and Peace* has about 43,000 lines. The full release of Debian version 4 for Linux [operating system] has about 283 million lines of code, which is thus about 6,500 times as big as *War and Peace* (Horn 2009).

Working with software is a solitary effort for one or a thousand individuals. What you see on the outside screen or in a printed report does not necessarily reflect what is going on inside. Bugs can lurk inside a system for years before surfacing when the triggering set of conditions finally occurs in the right sequence. How do you know what is going on? What is happening in the guts of the system? Where can things go wrong?

This book evolved from research into what long-term benefits some companies derived from changing the way they manage software development and maintenance. They did it to help answer some of these questions. In order to do that research, a model had to be developed which could be used in multiple types and sizes of companies to create a series of company profiles. These then could be compared to evaluate similarities and differences in company experiences.

The model developed was based on components from three sources. Therefore it is called the Multi-Model Assessment Instrument (MMAI) and it turned out to be very useful on its own merits. Models and standards abound; they usually have one of two formats: they look at many parts of the software process in great detail, or they look at a few parts of the software process, also in great detail. Using either type of model can be useful, particularly in a procurement, contracting or partnership situation. But a model like the MMAI may be more useful for an organisation at the very beginning of trying to understand how it manages software now and how it could do better. The MMAI covers a broad spectrum of issues at a basic level for a good overview of strengths and weaknesses.

Read the book, particularly the profiles of companies interviewed. See if you think using the MMAI could help your organisation understand its software development and/or maintenance practices better.

ACKNOWLEDGEMENTS

To give credit to all the people, groups and companies who made this possible would create a new document with many chapters going all the way back to thanking my first instructor in FORTRAN at Purdue University in June 1961. This book documents a history I lived as well as the current research and I thank the many people who made it all possible.

For this particular exercise in frustration, I have my advisor, Dr. Ita Richardson, to thank. She has suffered with me through sloughs of despair to bring us through to the other side.

This research was partially supported by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie). It is hard to overstate the value of support from Lero personnel, ranging from Professor Kevin Ryan through my graduate student companions and the administrative staff who have been there to help in diverse ways. Dr. Jim Buckley, Dr. Norah Power, Professor Eamonn Murphy and Professor Brian Fitzgerald all had helpful comments and suggestions during the research and Jim's thoughtful review of an earlier paper and detailed comments were particularly useful.

My husband, Joc, is approaching sainthood.

CHAPTER ONE

INTRODUCTION

1.1 Introduction to the research

Software problems have existed since the earliest days (O'Connell 1994, Pulford *et al.,* 1996). These problems have been met with many different solutions, including improved processes, other software methods, standards and improved skills as shown in Figure 1-1. Sometimes these solutions have been implemented sequentially, sometimes concurrently.

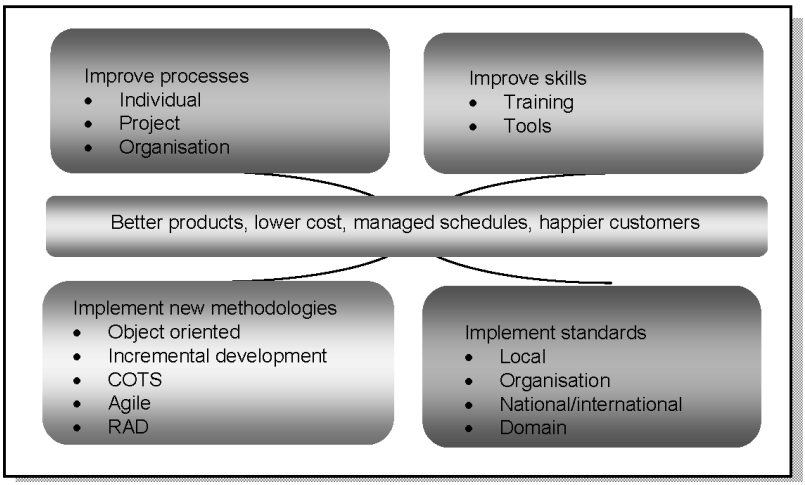

Figure 1-1 Software problem solutions over time

This volume reviews a range of software problems and solutions and explores how general patterns evolved as tools and the development experience have improved. The concept of software process improvement (SPI) has grown since it was originally proposed (Humphrey 1990) to become an environment where a range of SPI standards and models exist.

These have been used to help companies manage software development and maintenance more effectively, to provide an opportunity for recognition, to demonstrate capabilities and to deliver the products and services required.

The research presented was designed to:

1) Evaluate the commonalities and differences in selected standards and models
2) Adopt or develop an assessment model which could be used in companies that incorporated these different standards and processes for improvement
3) Use the chosen assessment model to evaluate the current status of companies which participated in SPI programmes sponsored by the European Commission from 1994 through to 1999; applying this model enabled improved validation of the usefulness of the model, and also determined a likely answer to the question, "Was SPI assistance useful in helping these companies improve their software processes?"

1.2 Structure of this book

Chapter two describes the software environment, starting with the general culture of software development and maintenance that makes it so difficult to manage. Specific problems including late deliveries, cost overruns on software projects and poor quality software, are all described, as are some of the solutions which were developed to improve these issues. General solutions, including various standards and models, were developed to change the entire culture of an organisation. This was followed by the idea of software process improvement—better processes lead to better products. Certification of an organisation's capability to provide the required goods and services has been driven and enabled by the evolution of all the major SPI standards, thereby leading to extended globalisation of the software market.

Chapter three describes SPI events in Ireland between 1994-1999, and the historical research undertaken in relation to them. Some short-term benefits of SPI as described in various reports and case studies are presented and programme differences outlined.

Chapter four looks at successes and failures in SPI from researchers in different locations and types of environments.

Chapter five outlines the environment for this research and the methodology followed.

In Chapter six, the development of a chosen model for cultural assessment, the Multi-Model Assessment Instrument (MMAI), is described.

Chapter seven shows profiles of the companies interviewed for this research, describing both their culture and improvement experiences.

Chapter eight presents the actual data collection and analysis. Extraction of the collated data and the correlations of data which are of particular interest are also discussed. Comparisons are made between the results of SPI in the literature and results from this research.

Chapter nine is a summary, with conclusions about the usefulness of the MMAI, company improvements and the objectives which were met. There is a review of the research process and its limitations, the contributions facilitated by this research, the post-research environment and some plans for the future.

The appendix shows the interview guide used to gather information.

CHAPTER TWO

THE SOFTWARE ENVIRONMENT

2.1 The culture of software development and maintenance

Software project management can be difficult due to the inherent nature of the product. The most common software-related problems identified include intangibility, complexity and volatility of requirements (Jurison 1999). A lack of understanding concerning the relationships between technology, people, the organisational culture and the general business environment all contribute to project failures (Mitev 2000). Personal/political reasons may also make people hope that a project fails (Rost 2004). Software managers are frequently not business people and vice-versa.

2.2 Problems in software development

2.2.1 Introduction

The losses from software failures can be expensive. Out of the many examples a few stand out. In Ireland, the Health Service Executive and other government departments spent over $190M on computer systems which overpaid staff, had security issues and other continuing problems including suspension before completion for some systems (Coulter 2005, Donnellan and Reid 2005, McGee 2008, Horn 2009). In the UK, J Sainsbury PLC abandoned its supply-chain management system after deployment for a loss of nearly $600M (Charette 2005). McDonald's Corporation cancelled the Innovate information-purchasing system after $170M was spent (Charette 2005). The literature is full of examples of money spent for little or no return (Flowers 1996, Labanyi 2005, Reid 2005, Horn 2009) to name a few.

2.2.2 Specific problems and solutions

Many reasons for these types of failures that have been identified include: late deliveries and cost overruns, poor software quality, inadequate value management and lack of support. For each of these problems, potential solutions are presented.

2.2.2.1 Late deliveries and cost overruns

Projects that overrun schedules and budgets, as described by Brooks (1975, reprinted in 1995) are improved through better project estimation and/or project management processes. Risk mitigation for a company is also sometimes expanded to include risk management techniques specific to software systems.

Project estimation

Description

Good project estimation consists of understanding the tasks which need to be done and the time it takes to do them. Where this knowledge is absent, projects tend to overrun their schedules and budgets and software products are inadequate or incorrect. For example, one of the principal mistakes made in project estimation was noted by Brooks (1995) as long ago as 1975: project duration does not depend linearly on staff-numbers and adding more people will not necessarily shorten the duration of a project.

Benefits

Good project estimation is important because it provides essential input to pricing, planning and budgeting processes and therefore ensures that everyone has a clear understanding of the price to be paid for a specific product or service.

Problems

Inconsistency in estimation may be a result of not using formal estimation models and a lack of training in using them (Grimstad and Jørgensen 2007, Gruschke and Jørgensen 2007). Schedule and resource estimation, planning to meet business goals and bidding for contracts can be confused with one another since they all have different goals. A single method of estimation is probably inadequate and inaccurate but this is frequently the way it is done, nevertheless (Jørgensen 2005). Technical people are asked to contribute to the estimation process but frequently in

business terms that they would normally not use (Putnam *et al.* 2005). Estimation frequently does not take into account team sizes or the impact of schedules, costs and quality constraints (Putnam *et al.* 2005). Of course estimation is no better than the knowledge of the estimators. Software requirements always change (Jones 2008). Changing requirements, a vague project description, or other issues can make estimation a very difficult task (Roetzheim 2005). An experienced member of one team said that on all projects more than 50% of the effort was a consequence of unexpected events (Jørgensen 2005).

Project management

Description

Regardless of their size, all projects exhibit certain common characteristics that distinguish them from other types of work. Projects are temporary, undertaken to provide a specific product or service, have a finite schedule, clear outputs and goals, a specific leader, a schedule and particular resource requirements (Clements *et al.* 2005, Jurison 1999). Projects are also carried out by teams (Jurison 1999).

The role of project management is expanding. The decreasing costs of equipment and communication, the increased costs of people and potential failures, the increased complexity of software, and the increased availability and capability of tools have led to profound changes in project management. Distributed project management is used to support the development of large, complex systems in many sectors (Nidiffer and Dolan 2005), one of these being global software development (GSD) as initiated in more than one country.

GSD contains challenges for project management that may already have existed at some level in other distributed development enterprises but have become more visible with GSD. Amongst these challenges is the need for better planning, the enhanced management of people, more efficient organisational structures, risk management, infrastructural processes, conflict management and more supportive team structures and organisations (Ebert 2006, Casey and Richardson 2006).

Decision support systems are evolving to accurately estimate dynamic project behaviour (Donzelli 2006). More specialised tools are being developed to marry the organisational chart with the precedence diagram for the dynamic re-assignment of resources or the assignation of additional capabilities which may be needed at certain times in the project (Rifkin 2000).

Benefits

Good project management enables one to reach the project objectives within time and in accordance with estimated cost and performance. These three variables are the critical project dimensions which require continual project management attention. More recently, managers have added a fourth constraint: good client relations. To manage the above constraints two additional factors are essential: visibility and commitment (Jurison 1999).

Problems

Good project management is difficult. People have different skills and personalities, with productivity studies showing substantial differences between individual developers. No single methodology can be applied to every project (Howard 2001). This means that the project manager needs good management skills and flexibility in how to apply them. Project management with GSD has its own set of problems, including: lack of shared values, poor understanding of requirements, inadequate management of shared artefacts, unclear work assignments and ownership, insufficient communication, inadequate management strategies, unknown legal requirements, a high employee turnover rate or cultural and linguistic difficulties (Ebert 2006, Casey and Richardson 2006, Sarma and van der Hoek 2006, Hsieh 2006).

One study examines the factors for success or failure. These include: problems with the project manager and the overall management of the team; poor information gathering, a lack of understanding for the management of requirements and/or changing requirements. Optimistic or inaccurate scheduling and poor estimation are also amongst the factors contributing to project failure (Verner and Evanco 2005).

Risk management

Description

The risk management process as described by Conrow (2005) is outlined in Figure 2-1.

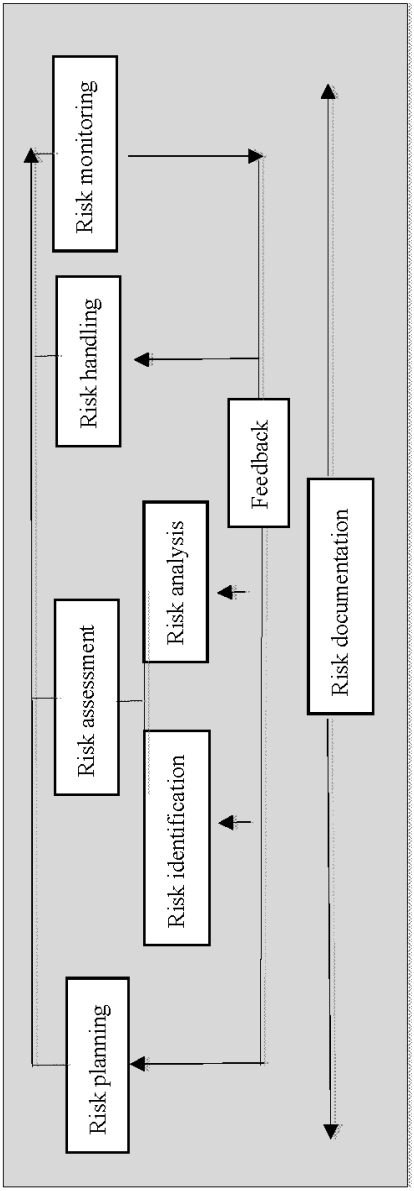

Figure 2-1 Risk management process

Succinctly, risk can be defined as a potential problem, event, hazard, threat, or situation with undesirable consequences (Hantos 2005).

Three major factors which influence project risk are: project size, project structure and experience with technology. The larger the project, the less structured it is and the less experienced the team are with the technology of the project, the greater the attendant risk. To counteract this, during the planning phase, the project manager needs to perform a realistic assessment of risks and develop a plan for controlling these same risks (Jurison 1999).

Benefits

In one sense, risk management is the most important activity for any company. If a company is not managing risk, what are its chances of even staying in business? In another sense, risk management is the umbrella under which everything else fits. A good risk management strategy should identify whether better project management, configuration management, project estimation, or other improvements are needed. Unfortunately, risk management seems to be both poorly understood and seldom practiced.

Problems

It is necessary to analyse the most significant risk drivers such as potential difficulties with schedule, technology, funding, integration, etc. Then develop risk scenarios so as to determine those events or trigger points which warn of any imminent risk (Holt 2005). However, it is not always so easy to determine what these risk drivers are, particularly on a new type of project.

For example, a risk assessment tool was used in the examination of 720 projects by senior IT managers in 60 large companies. Some of the findings were a surprise to the researchers because the top risk driver was one which is often neglected by management (use of inappropriate technology), while the least influential risk driver (requirements volatility) was one which managers often complain about the most. Six key risk drivers were identified, along with their relative importance to software project risk, and this, in turn, led to the introduction of a one-minute risk assessment tool that can be applied to improving software practice (Tiwana and Keil 2004).

Regardless of individual risks, risk has to be addressed from a system perspective in order to prevent: costly delays, increased stress for team members, development of a lesser product or even project failure (Holt 2005). For systems which incorporate large systems themselves, additional issues need to be understood, including such questions as multiple

stakeholders, long life cycles, integration risk as a result of complexity, and other issues which may or may not be important within smaller systems (Conrow 2005).

Another example is the special environment that is object-oriented software development, a process which carries its own set of risks. Boehm's top ten risks were re-evaluated in an object-oriented sense and shortfalls in external components, architecture, performance and quality were identified as being far more important than in other environments (Hantos 2005).

The conclusion reached here is that every organisation and project should be evaluated for the special risks associated with its unique attributes, rather than attempting to follow one set of recommendations for the entire software environment.

2.2.2.2 Poor software quality

Poor quality software has been rectified at several levels by learning to gather and manage requirements better, by testing quality within the software itself, and the use of quality management tactics.

Requirements elicitation and management

Description

To "elicit" means to draw forth, to bring out something's latent potential, (Brown 1993). So requirements elicitation is simply the gathering of requirements. As is expressed more broadly in ISO TR 15504 (1998) requirements elicitation means: *to gather, process and track evolving customer needs and requirements throughout the life of the software product and/or service so as to establish a requirements baseline that serves as the basis for defining the required software work products.* Requirements are elicited through interviews, questions, observations and various other techniques; no matter which process is used, some level of structure is preferred (Dieste *et al.* 2008).

One definition of requirements management is from the CMM, Paulk *et al.* (1993) *Requirements management is used to establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project.* Activities include the following: the software engineering group reviews the allocated requirements before they are incorporated into the software project; the software engineering group uses the allocated requirements as the basis for its software plans, work products, and activities; changes to

the allocated requirements are duly reviewed and incorporated into the software project.

High quality requirements of any type are adequate, unambiguous, consistent and verifiable (Glinz 2008). Functional requirements are those which define what the product *does*; these form most of the requirements as defined by the customer. Non-functional requirements are those that define *how* the product does the task; this includes quality, performance and maintenance requirements. Technical requirements are those requirements which define how the system is built—hardware, language, interfaces to other systems, etc. (Dekkers 2005). Requirements come from many sources including customers, users, other members of the organisation and other constraints due to hardware or system limitations.

Benefits

Requirements specifications define what the software is required to do. With well-managed requirements, developers can design the correct software, testers can test the software and compare the results with the requirements and programmers can write the correct code (ISEB 2002). Good requirements elicitation and management are necessary for good design and implementation, although it is not a guarantee of success on every occasion.

A computer system is designed and based on the requirements the developers have been given. Therefore, the more clearly the requirements are understood and expressed, the more likely the system will meet the customer's needs.

Problems

Project estimation is a formidable task from the beginning—especially before and during the requirement discovery process. Poor requirements lead to poor estimates and poor schedules. An estimate is only as good as its least reliable input variable (Dekkers 2005). One of the most common and best-known problems in project estimation is the failure to deal with requirements, which are about a complex real world and not simply a computer system (Jackson 2004). Part of that current complex real world is GSD, where the requirements have to be tasked and managed across cultural, time-zone and organisational boundaries (Damian 2006). Additionally, there are usually more requirements than can be implemented with the time and resources available (Karlsson and Ryan 1997). A successful process will incorporate a risk-based, value-oriented strategy in order to prioritise the requirements and mitigate the risks (Glinz 2008, Feather *et al.* 2008).

Requirements change, particularly during long projects. If the changes are not documented and well-managed, systems will be designed and developed according to the wrong requirements. Additionally, it is difficult to avoid defects in functional requirements. After a major failure, it is usually easy to see where more attention and care were needed, but this is not so obvious while the system is being developed (Jackson 2004). More specific problems may include: users cannot express their requirements unambiguously, users cannot express their requirements completely or developers do not fully understand the business (ISEB 2002).

Testing

Description

A test is a controlled exercise involving an object under test, a definition of the environment, a definition of the inputs, a definition of expected outputs or results. When a test is performed you get an actual output or result and a determination as to whether the result is correct. Common stages of testing are: component or unit testing, link testing (integration testing of a few components), functional and non-functional system testing, integration testing of major subsystems and the whole system, and user acceptance testing. Regression testing tests the rest of the system which has not been intentionally changed to ensure it still works correctly (ISEB 2002).

Benefits

A human being can make an error which introduces a fault to a program, thereby causing possible failure of the software. Testing helps find the faults before they translate into failures. Successful testing can save significant effort and increase product quality, thereby lowering maintenance costs (Juristo *et al.* 2006).

Problems

Component or unit testing is normally done by the developer but is constrained by what the developer knows regarding testing. For example, frequently developers do not consider full structural coverage (Runeson 2006). Despite obvious benefits, testing practices in industry generally are not as sophisticated or effective as they should be (Juristo *et al.* 2006). The level of effort required for testing has to be adjusted to the size of the job and the organisation, although it is often difficult to know the scope (Menzies and Cukic 2000).

Also, testing can be a very expensive way to find potential problems, if that is the sole method employed.

Quality management

Description

Quality management has taken many forms over the years, but two particularly stand out. ISO 9001 is a quality management standard used throughout the world. Total Quality Management (TQM) was used in Japan in the 1970s and became popular in the US during the 1980s (Dellana and Wiebe 1992). Four essentials of TQM are: top management's direct involvement in the delivery of quality, customer orientation, company-wide participation in quality, and the use of systematic methods to resolve quality problems (Bagchi 1997). Both TQM and ISO 9001 are designed to change the culture, attitude and organisation of a company that seeks to provide customers with quality products and services which satisfy their needs (Lee and Roberts 1997). Both require management commitment to quality and changes throughout the organisation to support this attitude. Activities are included which increase the level of confidence in quality from customer focus in designing the business processes down to software tests and inspections as added to the software life cycle.

Benefits

TQM is a management philosophy that seeks to integrate functions and focus on meeting customer needs. ISO 9001 is also focused on meeting customer needs but with specific processes defined. The premise is that high quality leads to high productivity and shorter schedules (Jones 2008).

Problems

Quality, like beauty, is in the eyes of the beholder. Developing a notion of quality that everyone understands, whether or not they agree with it, has proven difficult. For example, TQM has as a basic tenet that all processes are continually improved through process improvement and statistical process control (Biehl 2004). Not everyone sees the same benefit or the same cost through the implementation of TQM, however. The business owner is trying to strike a cost/benefit balance that will enable the business to make a profit and provide products and services the customer(s) will buy and use. Different customers are likely to have different ideas of what quality is. The users may have a different idea about how the system operates and whether it has quality or usability. TQM requires non-stop effort from everyone in the company. However, this practice can sometimes