Faculty of Computing,
Engineering and Technology

IMPROVING THE QUALITY OF COMMERCIAL SOFTWARE ENGINEERING THROUGH PROJECT PLANNING AND MANAGEMENT IN ALIGNMENT WITH CORPORATE STRATEGIES

MSc Computer Science

Gregor Dominik Porzel

gregor.porzel@gmail.com

059 649 62

A thesis submitted in partial fulfilment of the requirements of Staffordshire University for the degree of Master of Science.

March 2009

Supervised by David Thomas

Abstract

This work is concerned with the question why software engineering is still problematic in many cases compared to other knowledge areas. The aim is to develop a method which can improve planning and management of commercial software engineering projects. To achieve this goal, this report investigates the current situation of the software engineering discipline and describes detailed research into potential problem sources. The findings of the literature review lead to the question whether recognition and consideration of implicit corporate business factors and integration of these factors into planning for commercial SE projects may be a potential key to successful commercial software engineering projects.

Analysis of shortcomings in current planning methods and investigation into planning for implicit corporate business factors result in the production of two artefacts: the Commercial Software Engineering Planning Framework and the supporting software prototype *IntelliPlan.*

This work conducts mainly secondary research, but comprises primary research in validating the deliverables: the functional and semantic tests have shown that the concept might represent a promising means to significantly improve the quality of commercial software engineering.

Table of Contents

List of Figures

List of Tables

Acknowledgements

The author is grateful to Dave, who provided guidance for the project not only in a formal way but through mentoring the author concerning awareness, consideration, compassion, enthusiasm and life in general.

Also, he is thankful for the support of Andreas, Harald, Mani and Schorschi who are loyal friends. The author expresses many thanks to his fellow students Ali and Sebastian who helped to spark ideas through discussions.

The catering employees of Staffordshire University need to be mentioned at this point, as they not only managed to take the focus away from technical topics for a few minutes (in order for the author to return to research with a refreshed mind), but also provided delicious coffee.

This dissertation is dedicated to my parents Theodor and Christine Porzel.

Chapter One

1 Introduction

This chapter introduces the research on how the quality of commercial software engineering projects can be improved through aligning the project planning and management with corporate strategies. The structure of the work allows the reader to learn about the topic of software engineering and related areas while progressing through the seven chapters.

The following sections convey the context and scope of the topic as well as an overview of contributions to the body of knowledge in the field of software engineering. In addition, this chapter outlines the context of the academic community with respect to the findings of this work. Aim and objectives of the work undertaken are provided to form the basis for a list of deliverables which are produced in order to provide a sound answer to the research question stated in this chapter.

Further sections comprise a brief description of the applied research methodology, ethical considerations, a statement concerning the intellectual challenge and an overview of following chapters. This introduction chapter closes with the time plan for the research project.

1.1 Key Contributions to the Body of Knowledge

1.1.1 Primary Contributions

A major contribution of the research constitutes findings concerning the problematic of existing software engineering project planning methods, concepts and tools. Furthermore, consolidation of factors which might influence commercial software engineering projects may add value to the body of knowledge. The creation of a framework based on these findings could address the issues with commercial software engineering projects elicited in the research.

1.1.2 Secondary Contributions

The literature review provides a sound and concise overview of how the software engineering discipline emerged. Furthermore, potential reasons for prevailing issues with software projects as well as possible solutions are investigated through analysing, comparing and contrasting predominantly acknowledged scholarly sources.

Also, a suitable prototype was developed to support the framework and to enable efficient validation of the theory. This adds value to the body of knowledge in that accurate validation can be crucial when formulating an answer to the research question.

1.2 The Work in the Context of the Academic Community

Software engineering is a topic of fairly wide range. For example, the body of knowledge contains highly specific technical research as well as rather generalised methodologies for project management. The research in this work is aimed at the planning of software projects. However, in order to improve project planning, potential problem sources faced by software engineering projects are investigated. This in turn leads the work having to comprise not only software engineering-specific methods and models, but also to include influencing factors from a higher vantage point.

Therefore, albeit the research is focused on the creation of a suitable planning concept for commercial software engineering, many findings may be utilised by other fields as e.g. real time- or research and development projects.

1.3 Aim and Objectives

The aim of this dissertation is to develop a method which can significantly improve the planning and management of commercial software engineering projects.

The objectives of the work to support achievement of the aim are as follows:

- Investigation of the current situation of the software engineering discipline.
- Research into potential issues of software engineering from academic- and practical viewpoints.
- Comparison of long-established and accredited engineering disciplines with software engineering.
- Investigation into adjacent, potentially influencing areas as e.g. corporate finance and business management: this research is to elicit potential impact of external project conditions on software engineering.
- Based on the research findings: creation of a framework to enhance and improve planning and management of commercial software engineering projects.
- Development of a project planning and management software application prototype to support the framework and improve the accuracy of the validation.
- Reasonable functional tests and semantic validation of the deliverables in collaboration with one academic expert and two industry experts using a simulated project scenario.
- Provision of a sound conclusion including reflective summary, recommendations for further research and development as well as description of a potential future outlook.

1.4 Research Question

Is recognition and consideration of implicit corporate business factors and integration of these factors into planning for commercial SE projects a potential key to successful software engineering projects?

1.5 Deliverables

- A report containing
  - research progress (literature review and research chapters)
  - development of artefacts (framework and prototype)
  - presentation and documentation of the prototype
  - validation of deliverables
  - conclusion.
- A disc containing
  - digital version of the report in two different file formats
  - the source code for the software prototype including the solution file and project files for the usage with Microsoft Visual Studio 2008
  - the compiled version of the software prototype in the form of an installation routine.

1.6 Research Methodology

The literature review and analysis of existing project planning tools constitute secondary research. Where appropriate, research chapters other than the literature review contain references to sources from the literature, too.

Primary research was conducted to validate the deliverables: the opinions and ideas expressed by academic and industry experts contribute not only to the validation chapter, but were also considered when formulating the conclusion.

1.7 Ethical Considerations

This project complies with the ethical regulations of Staffordshire University. In order to insure compliance with these regulations, the following steps were applied in a disciplined manner during validation of the project:

- The participants were informed about the validation procedure in advance and they were asked whether they fully understand the procedure.
- It was made clear to the participants that their participation is voluntary.

- The participants were informed that the validation is not observational and is conducted in a cooperative manner.
- The participants were informed that they can withdraw from the validation at any time and for any reason.
- Participants were given the option to omit questions they do not wish to provide an answer for.
- It was made clear to participants that their personal information and collected data is being treated with full confidentiality. Furthermore, they were informed that published information of any kind will not be identifiable as theirs.
- Participants were informed that they have the option to be debriefed regarding the outcome of the research project they contributed towards.
- No participants were misled in any way.

The steps above fully comply with the *Staffordshire University Fast-Track Ethical Approval Form* for students.

The ethical consent form which was handed out to participants can be found in Appendix D.

1.8 Intellectual challenge

Firstly, as mentioned above, software engineering comprises a vast amount of knowledge areas. Thus, revision and critical evaluation of the existing software engineering body of knowledge and related fields pose an intellectual challenge, because research areas need to be chosen with care and justifiable focus. Furthermore, the literature concerning external factors potentially influencing software engineering projects such as legal aspects is to be explored in the same manner.

Secondly, findings of the literature review have to be sound enough in order to form the basis for further research chapters. This also applies to the research chapters as they have to provide sufficient relevant input for production of the artefacts (framework and software prototype).

Thirdly, the development of the software application to support the framework and aid in validation presents technological challenges: the prototype shall possess a user-friendly graphical interface as well as needs to reflect the theoretical framework in a practical way. In addition, functional tests are to be conducted to insure the application has a low error rate and is ready for semantic validation in collaboration with experts.

Fourthly, validation of the deliverables is to be integrated into the work without bias or modification and interpreted in a sound manner. This insures that the findings can be used as an information source to form a meaningful conclusion.

1.9 Chapter Overview

The research, artefact and validation chapters begin by briefly introducing the findings of the respective previous chapter. This enables the reader to progress through the work in a random fashion. However, readers who decide to follow the chapters in numerical order may find that their knowledge and understanding of software engineering and its role in companies unfolds beyond mere research findings. The intention behind structuring the work this way is to allow the reader to learn how the solution proposed in chapter five came into being.

The literature investigation starts in chapter two, which forms the basis for consecutive research chapters. This literature review sheds light on how and why the discipline might have emerged and evolved the way it did. The chapter describes prevailing issues and potential reasons for these difficulties. Furthermore, it elicits factors which may be the key to successful software engineering projects.

In chapter three, analysis concerning planning for commercial software engineering projects is conducted. This includes a presentation of existing project planning tools as well as a discussion on the suitability of these common approaches.

Chapter four investigates potential factors which may be of importance to project success. These factors are implicit, but could exert vital influence on commercial software engineering projects. In addition, the chapter discusses the topic of quality from an unconventional new perspective.

Chapter five contains the description of artefact deliverables, which represent the practical results of the research. These include the Commercial Software Engineering Planning Framework and the software application prototype IntelliPlan.

Evaluation of the deliverables is outlined in chapter six and comprises functional tests of the software prototype as well as semantic validation through an academic expert and two industry experts.

Chapter seven provides a conclusion reflecting on research findings and deliverables. In addition, the final chapter contains recommendations for further research and development as well as a future outlook.

1.10 Research Project Time Plan

Activity	Number of weeks
Chapter One: Introduction	1.5
Chapter Two: Literature Review – Consolidating Software Engineering	4
Chapter Three: Planning for Successful Commercial Software Engineering (SE) Projects	2
Chapter Four: Implicit Corporate Business Factors	2
Chapter Five: Planning for High Quality in Commercial SE Projects	3.5
Chapter Six: Testing and Validation	5
Chapter Seven: Conclusion and Future Outlook	3
References, Bibliography and Abbreviations	1
Abstract and Review	0.5
Contingency Time (Including further Review)	1.5
Total	**24**

Table 1 – Research Project Time Plan

Chapter Two

Several published works over the last decades describe difficulties when engineering principles were applied to the development of software. The body of knowledge leads to believe that these issues in turn could be the cause of delivery delays, budget overruns and failures of software projects. While some scholarly sources propose potential solutions, others deny the existence of an ad hoc solution to the difficulties one encounters when developing the complex product that is software.

2 Literature Review: Consolidating Software Engineering

2.1 Introduction

The purpose of this literature review is to shed light on the current situation of the software engineering (SE) discipline. The chapter begins with a brief historic overview of SE and includes essential knowledge areas such as software process models and development methods. The work continues to explore problem areas within SE in contrast with other engineering disciplines and potential solutions to these issues.

Also, this chapter provides investigation into SE from a rather unconventional perspective: few authors seek the root of the problems outside SE models as e.g. in common misconceptions surrounding the discipline. One example is the recognition that the term *software engineering* in itself might be ambiguous, potentially leading to the application of inefficient methods as shown in subsection 2.8.1 of this chapter.

This chapter closes with a review of sources concerning the management of software projects as this forms the basis for subsequent research- and investigation chapters.

2.2 Software Engineering: The Emergence of a New Discipline

The software engineering discipline was partially inspired by a situation which occurred during the 1960s, denoted as the "software crisis". This particular crisis "(…) resulted directly from the introduction of third generation computer hardware" (Sommerville 1993, p. 3). More powerful hardware allowed for software systems of hitherto unprecedented functionality and size. "It was clear that better organized methods and more disciplined practices were needed to scale up to the increasingly large projects and products that were being commissioned" (Boehm 2006, p. 14).

There were two main reasons to call for more structured procedures in software development:
firstly, "(…) existing methods of software development which worked fine with small systems were not good enough for large systems. Techniques applicable to small systems could not be scaled up" (Sommerville 1993, p.3).

> *There are many areas where there is no such thing as a crisis — sort routines, payroll applications, for example. It is large systems that are encountering great difficulties. We should not expect the production of such systems to be easy.*
>
> (Naur & Randell 1969, p. 9)

(The implications of developing large applications rather than smaller programs are elucidated in more detail in subsection 2.3.1).

Secondly, structured development turned out to be inevitable, for during the 1950s "(...) the prevailing thesis was (...) *Engineer software like you engineer hardware.*" (Boehm 2006, p.13). Software however can be modified with much less effort than it is the case with hardware, which "(...) led many people and organizations to adopt a *code and fix*[1] approach to software development, as compared to the exhaustive Critical Design Reviews that hardware engineers performed (...)" (Boehm 2006, p.13). Brooks (1995, p. 218) cites Coqui (1989) who identifies that "The driving force to use Software Engineering principles in software production was the fear of major accidents that might be caused by having uncontrollable artists responsible for the development of ever more complex systems".

Hence, there was an apparent need for regulation to enable large-scale high quality software development – preferably by *outlining a new engineering discipline.*

In 1968, the infamous NATO Software Engineering Conference was held at a time when the number of installed computers in Europe did not significantly exceed 10,000 machines. It was during this conference when "(...) the term [software engineering] was coined in 1968 by F.L. Bauer of the Technological University of Munich (...)" (Dijkstra 2007, p.6).
"The Conference was to shed further light on the many current problems in software engineering, and also to discuss possible techniques, methods and developments which might lead to their solution" (Naur & Randell 1969, p. 8). As early as 1969, professionals in the field must have anticipated the emergence of a new discipline that could impact many aspects of peoples' lives: "One of the major motivations for the organizing of the conference was an awareness of the rapidly increasing importance of computer software systems in many activities of society" (Naur and Randell 1969, p. 9). Since the time when the conference took place, numerous methodologies[2], methods, process models and paradigms were added to the SE body of knowledge. The following section is to provide a guide to some of these widely acknowledged software development concepts.

---

[1] *Code and fix* refers to modification of program code in a rather unstructured manner.

[2] With regard to software engineering, *methodologies* can be seen as a means of combining SE models and methods, depending on project conditions and -attributes.

2.3 Software Process Models and Methods

2.3.1 What Types of Programs require a Structured Development Approach?

The development of large-scale applications is different from small computer programs. Brooks (1995, p. 4) uses the following example: "One occasionally reads newspaper accounts of how two programmers in a remodelled garage have build an important program that surpasses the best efforts of large teams". He continues to explain that such a program is "(…) complete in itself, ready to be run by the author on the system on which it was developed" (Brooks 1995, p. 4), but also argues that the program can be "(…) converted into a more useful, but more costly, object" (Brooks, 1995, p. 5). Figure 1 illustrates several options when developing a computer program. Moving across either horizontal- or vertical boundaries causes cost estimations to triple accordingly:

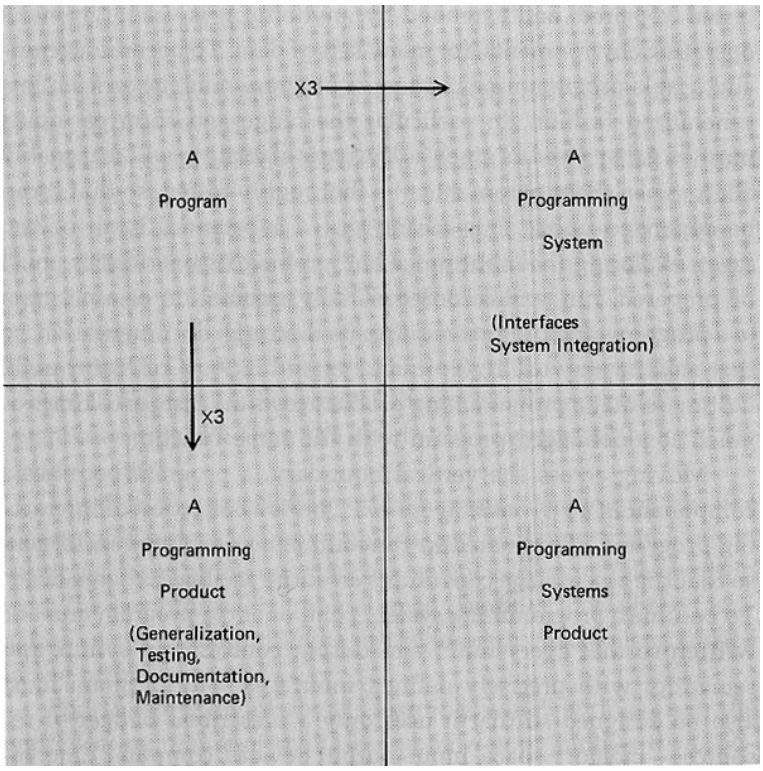

Figure 1 – Evolution of the Programming Systems Product (Brooks 1995, p. 5)

According to Brooks (1995), the costs are nine times as much when producing a *Programming Systems Product (PSP)* in comparison to production of a simple program. However, in the long run, more expensive alternatives can yield advantages, as for example low maintenance costs through sound documentation and better adaptability to the target environment through generalisation. Also, in case a customer requests changes of certain requirements, clearly defined interfaces can contribute to accelerated implementation of such changes. Therefore, *PSPs* can turn out to be the more cost effective solution in the long run.

2.3.2 The Reasoning behind SE Models and Methods

In order to ensure a high quality standard for software products, the development of *PSPs* as described above needs to follow certain guidelines. Some of these guidelines can be found in software process models. Also, regulation through models can aid in preventing usage of the *code and fix* approach introduced in 2.2. This approach is one of the main causes of high software maintenance costs, because it usually lacks documentation: even experts who are fluent in the respective programming language may require some additional time to understand the behaviour of the program before being able to change the source code[3] of the application without a high risk of introducing errors. The literature sometimes refers to this type of code as "spaghetti code".
The consequence of engineering software in a laissez faire manner can pose serious quality problems as identified by Boehm (1988):

1. Due to an absence of a design phase prior to coding, the program will become increasingly unstructured during maintenance. This in turn will make subsequent corrections more expensive.
2. The software might be built *right* from a technical point of view, but the *wrong* software may be developed from the perspective of the user. Hence, the finished application may fail to meet the requirements.
3. Fixing code produced using this type of approach can impose unnecessary costs due to "(...) poor preparation for testing and modification."
   (Boehm 1988, p. 62).

Hence, sound approaches to SE are required to improve the ability to control, direct and manage a software project and improve the quality of the resulting products.

With regard to the potential future of SE, vastly increasing size of programs will make it more and more necessary to stay in charge of complexity: "Current trends are leading us to systems of unthinkable scale in not only lines of code, but in the amount of data stored, accessed, manipulated, and refined; the number of connections and interdependencies (...) and the sheer number of people involved in some way (Fraser et al 2007, pp. 1028-1029).
Boehm (2008, p. 38) is of the opinion that software is crucial for collaboration and communication in the 21st century and claims that "The world is going to need all the capable software engineers or hardware and human factors engineers that understand software that it can find".

[3] *Source code:* This term refers to the editable and human-readable code of a computer program which has not yet undergone compilation (i.e. the automated translation into machine-oriented code).

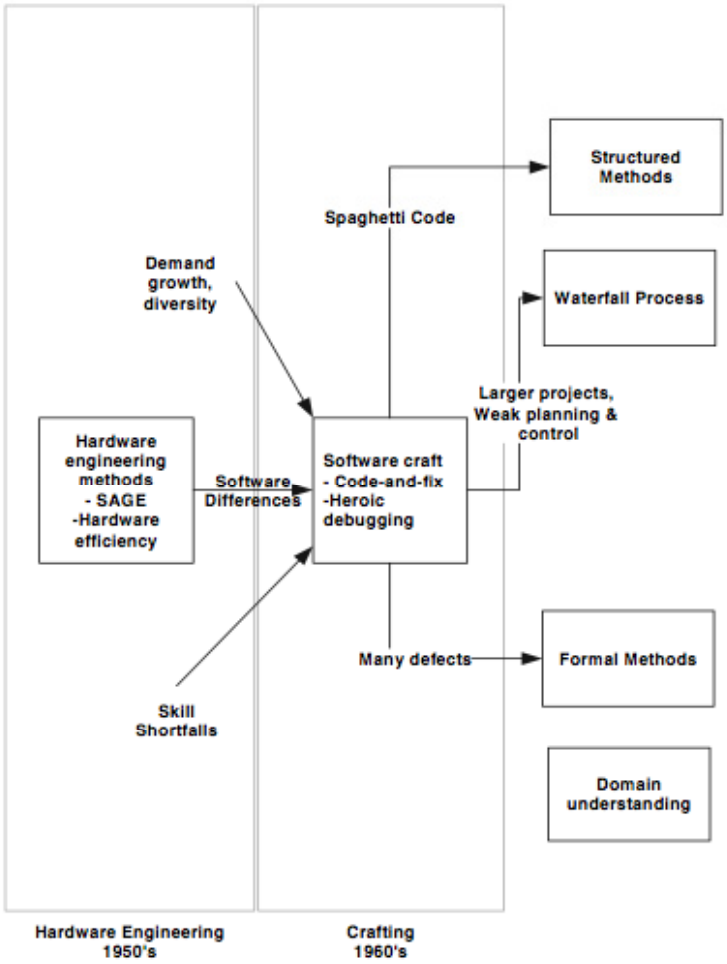

Figure 2 – Software Engineering Trends through the 1970s (Boehm 2006, p. 14)

As depicted in figure 2, the deficient methods of "crafting" software during the 1960s resulted in ideas, which – decades later – form an integral part of planning for software. One of these notions is the *waterfall model:*

2.3.3 The Waterfall Model

According to Boehm (1988), the waterfall approach is especially suitable for some specific types of software such as compilers or secure operating systems. However, Royce (1970), who introduced the model (while refraining from providing an actual name for the model), states his view: "(...) the implementation [of the waterfall model] is risky and invites failure" (Royce 1970, p. 329). In addition, he provides five steps that "(...) I feel necessary to transform a risky development process into one that will provide the desired product" (Royce 1970, p. 335).

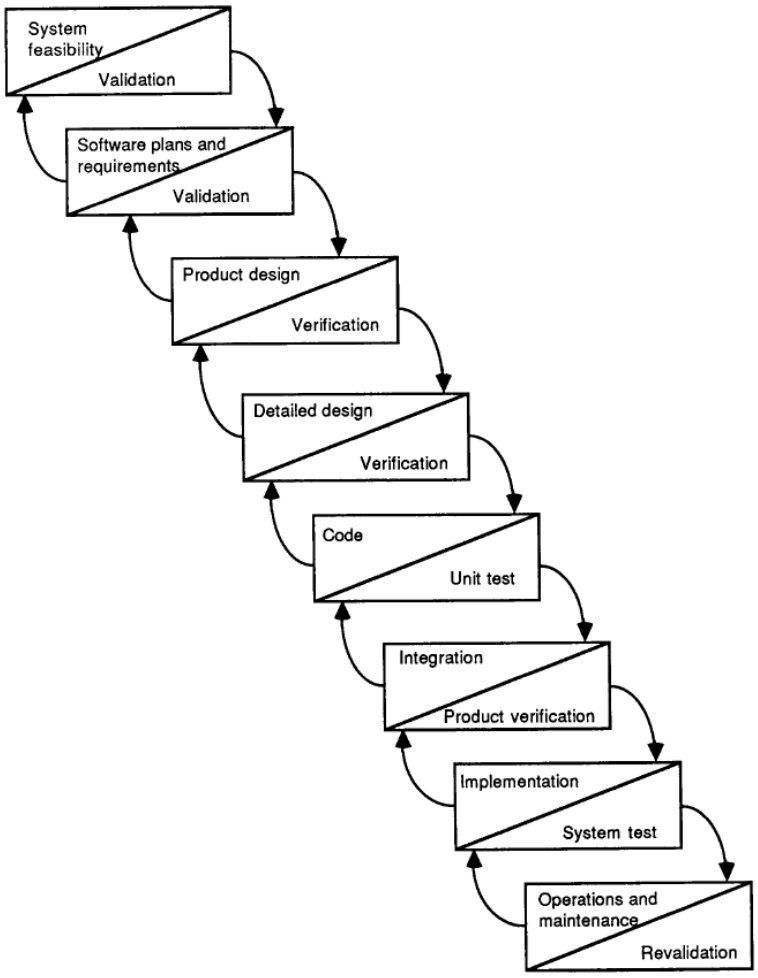

Figure 3 – The Waterfall Model of the Software Life Cycle (Boehm 1988, p. 62)

Boehm (1988, p. 63) concedes that the waterfall model "(...) helped to eliminate many difficulties previously encountered on software projects". However, he also admits fundamental issues of this concept: "A primary source of difficulty with the waterfall model has been its emphasis on fully elaborated documents as completion criteria for early requirements and design phases" (Boehm 1988, p. 63). He states that many projects comprised "(...) elaborate specifications of poorly understood user interfaces (...) followed by the design and development of large quantities of unusable code" (Boehm 1988, p. 63).

Sometimes users can be more precise in describing what they do not want rather than actual requirements. However, often users may find it difficult to provide this information until they actually work with the software (or a prototype). For example, when using the waterfall approach to conduct a software project, small changes to the software specification can mean for the entire project to be set back to the stages of requirements- and design. To make things worse, this could be the case even if such changes would only affect a small component.

Therefore, problems with the model might predominantly originate from the rigid and stepwise completion criterion, because "(...) there is only one entry and one planning session: the beginning of the project (...)" (Bergström & Råberg 2003, p. 36). Sommer-

ville (1993, p. 6) supports Boehm's view by stating that "(...) the realities of software development did not accord with the activities identified in the model".

2.3.4 The Spiral Model

The spiral model addresses the shortcomings of the waterfall model in that it explicitly recognises risk as an important factor for success in software projects (Sommerville 2007). The model was originally introduced by Boehm (1988, p. 71): "The risk-driven nature of the spiral model is more adaptable to the full range of software project situations than are the primarily document-driven approaches such as the waterfall model or the primarily code-driven approaches such as evolutionary development. It is particularly applicable to very large, complex, ambitious software systems".

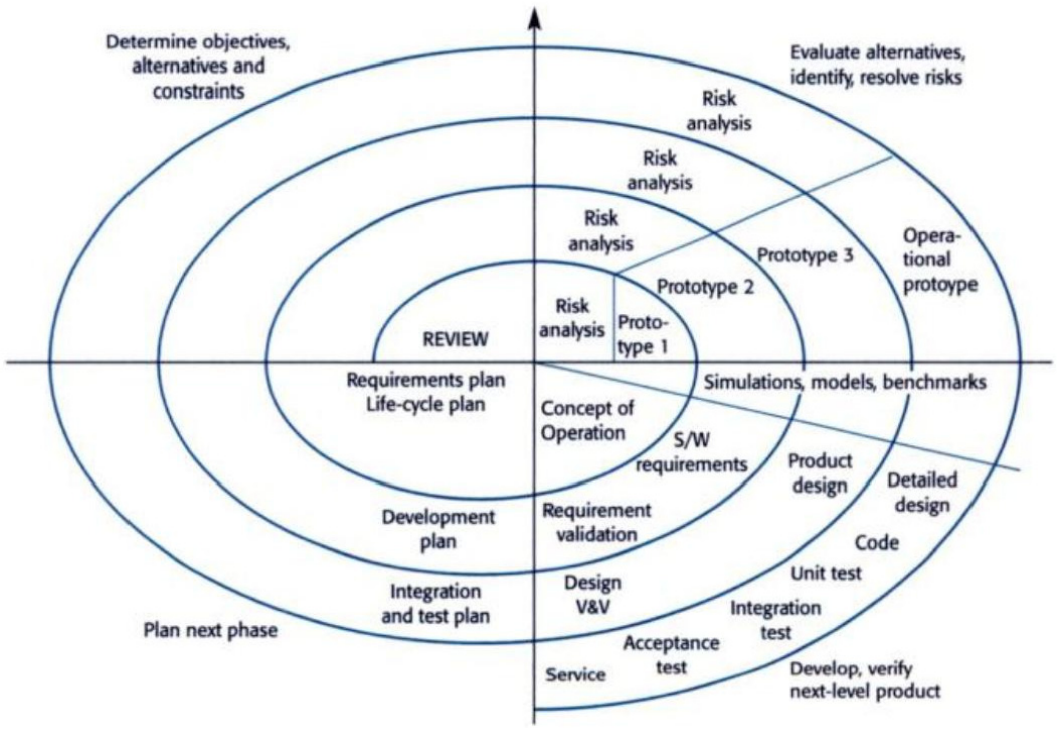

Figure 4 – The Spiral Model (Sommerville 2007, p. 74)

> *Let us turn to nature and study complexity in living things, instead of just the dead works of man. Here we find constructs whose complexities thrill us with awe. The brain alone is intricate beyond mapping, powerful beyond imitation, rich in diversity, self-protecting, and self-renewing.* ***The secret is that it is grown, not built.***
> *So it must be with our software systems.*
>
> (Brooks 1987, p. 18)

2.3.5 Rational Unified Process

The major advantage of the spiral model is that its incremental approach contains risk throughout the project. The Rational Unified Process (RUP) model also emphasises risk reduction in that "Risks need to be identified early in the project, along with their

potential impacts and possible alternatives to eliminate or reduce them" (Bergström & Råberg 2003, p. 38).

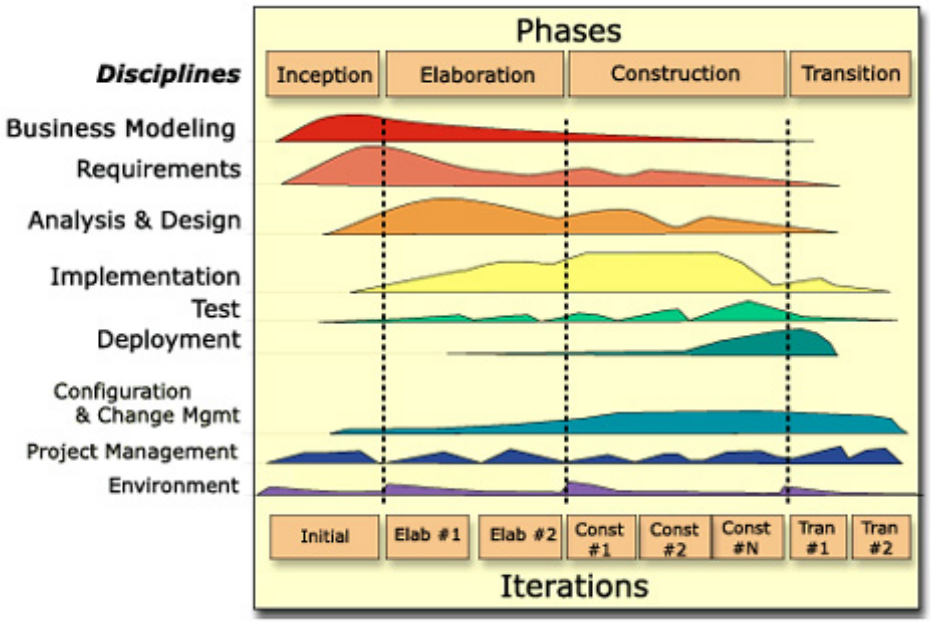

Figure 5 – The Rational Unified Process Model (Franklin 2005)

Iterations are crucial to the concept of RUP, because they enable the traversing of all stages for several times during the project. This in turn results in one increment[4] per iteration (see figure 6).

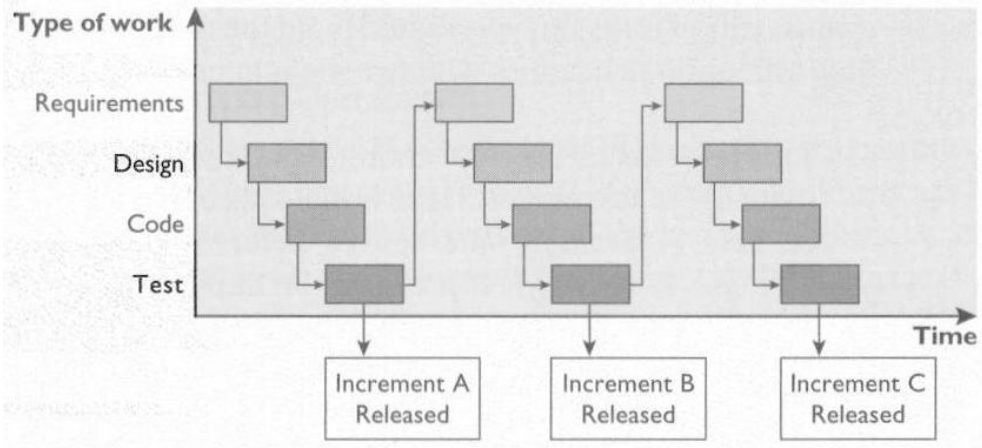

Figure 6 – Iterations and Increments (Bergström & Råberg 2003, p. 37)

Ambler et al (2005) describes RUP as a flexible framework structure from which a bespoke internal SE process can be created. Furthermore, they state that it is a success-

[4] Whereas iterations are measured in *time,* increments are measured in *progress* during development. For example, progress can be expressed in lines of code or functionality (Bergström & Råberg, 2003).

fully proven and tested concept. Information concerning an extended version (Enterprise Unified Process) of RUP can be found in Ambler et al (2005).

2.3.6 Further Approaches of Models, Methods and Paradigms

2.3.6.1 V-model

A modified version of the waterfall model is the V-model: "This model included validation and verification[5] processes by associating testing activities with the analysis and design phases" (Deek et al 2005, p. 13). Using this model, test specifications are to be written during relatively early stages in the project. This in turn ensures that "(...) earlier stages, such as requirements, high-level design, and low-level design, were properly accounted for in the later compliance stages of acceptance (...) integration testing and unit testing" (Deek et al 2005, p. 13).

2.3.6.2 Structured Systems Analysis and Design Method (SSADM)

In the UK (United Kingdom), the Structured Systems Analysis and Design Method (SSADM) has been used by the government for computing projects since it was launched in 1981 (Eva 1992). SSADM comprises the production of sequenced modules, which is an approach that at first glance may resemble the waterfall approach. "However, each Module is a self-contained set of activities, and must be managed as a discrete project" (Eva 1992, p. 19).

2.3.6.3 eXtreme Programming (XP)

Rather radical approaches to software projects with less emphasis on planning are sometimes being referred to as agile concepts. Sommerville (2007) identifies XP as one of the most widely used agile methods. The basic idea is to discard rigid patterns which may prevent SE teams from producing high quality code (Beck & Andres 2004). Beside other techniques, this may be achieved through utilising incremental development, pair programming[6] and strong customer involvement (Sommerville 2007). However, "The customer may feel that providing the requirements was enough of a contribution (...)" (Sommerville 2007). Increased involvement of client staff can become costly not only for the customer, but also for the supplier of the software as some clients could demand a price reduction in exchange for their participation in the development.

2.3.6.4 Rapid Application Development (RAD)

Rapid Application Development (RAD) is mainly useful for developing e.g. database applications which may have no out-of-the-ordinary requirements. "(...) there is a great deal of commonality across business applications (...). In essence, these applications

---

[5] Whereas *verification* addresses whether a system is built correctly and according to plan, *validation* addresses whether the "right" system was built – i.e. if the requirements were implemented correctly (IEEE 610.12 1990 cited in Deek et al 2005).

[6] *Pair programming:* developers work in pairs to insure that e.g. the work is double-checked and that the best solutions are being elaborated via mutual support (Sommerville 2006).

are often concerned with updating a database and producing reports (...). Standard forms are used for input and output" (Sommerville 2007, pp. 405-406). Kerr & Hunter (1994) state that RAD allows the delivery of systems with much higher quality than traditional ones while reducing the time for development. However, such generalised statements with respect to software development might be unrealistic, because it could be the case that some SE methods cannot be applied to certain types of software. The disadvantage of RAD lies in the restriction of scale: large systems require large teams, which are in turn more difficult to organise using this approach (Sommerville 2007). Also, implementation and integration of non-standard components can turn out to be costly as "There is no explicit system architecture and there are often complex dependencies between parts of the system (...)" (Sommerville 2007, p. 407).

2.3.7 Which Approach is the "Right One"?

This question may be raised frequently in situations where different concepts to achieve a certain goal are presented. One might recognise that it should be formulated more precisely: which approach is the most suitable one for the respective project and corporate environment.
On the one hand, small and medium enterprises (SME) may prefer agile methods which produce little managerial overhead for small projects. Vast undertakings of large organisations on the other hand, can involve up to thousands of stakeholders[7]. This in turn can trigger the demand for a more structured- but rigid approach in order to stay in charge of the complexity.

It needs to be emphasised at this point that successful usage of highly agile methods relies strongly on the competencies of the individuals as well as on the team as a whole including its leaders. The less SE experience is present in a team, the stronger the demand for regulation becomes. This regulation can help to insure that the deliverables meet the requirements and the project stays within time- and budget constraints.

[7] In general, a *stakeholder* is a person who has an "interest" in the project. This can be a programmer, the customer or internal departments as, for example, marketing and human resources.

2.4 Difficulties Faced by the Discipline and Potential Solutions

van Genuchten (1991, p. 582) describes a survey concerning software development projects, which was conducted by Jenkins et al (1984). 23 major U.S. (United States) corporations were analysed and it was found that "The average schedule overrun was 22%." and "(...) the cost and schedule overruns seem to be uniformly distributed among large, medium, and small projects".

With regard to IT (Information Technology) projects in general, recent articles demonstrate that problems often prevail to date: "Everyone knows IT projects don't work. Twenty years ago the Standish Group produced its shocking conclusion that 80 per cent of such projects failed. Our own research, with Harvey Nash, showed little improvement – a 63 per cent failure rate. Over the past two decades pretty much every other piece of research has reached similar conclusions" (Elton 2008, p. 2).

In his book *The Mythical Man-Month,* Brooks (1995 reprint of 1975 edition) describes large-system programming as problematic, stating that few were able to meet goals, schedules and budget restrictions.

"Unfortunately industry experience shows that such projects to develop and install large software systems are on average 50% late, with corresponding over-expenditure and serious 'teething problems' even after delivery (...)" (Grimson & Kugler 2000, p. 541). Decades before this statement was made, Naur & Randell (1969, p. 10) have provided the following appraisal: "There was general agreement that 'software engineering' is in a very rudimentary stage of development as compared with the established branches of engineering".

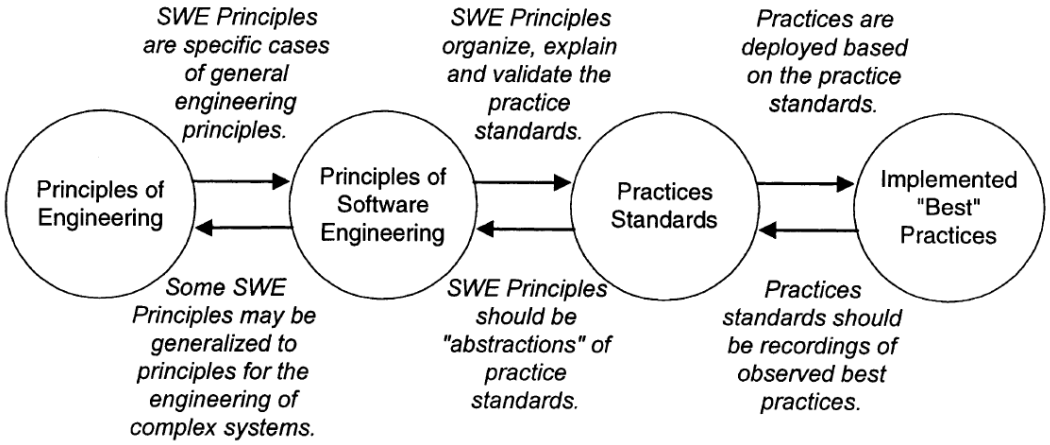

Figure 7 – Relationship between Principles and Practice (Bourque et al 2002, p. 61)

The idea of applying engineering principles to the "construction" of software seems appropriate at first, but might have some deeply-rooted flaws as shown in this section. In the following, issues with SE as well as potential solutions to these problems are discussed.

2.4.1 Individual Tasks and Technology

Educational measures help to exemplify the difficulties with SE originating from unforeseen events: training was conducted at Loughborough University; modelled after a similar course in a corporate environment. Dawson (2000) describes the training in his paper *"Twenty Dirty Tricks to Train Software Engineers"*. The goal was to improve preparation of students for the reality of working-life through simulation of real-world conditions. Among further ideas, the following concepts were included:

- Simulation of customer behaviour (e.g. frequently changing or conflicting requirements)
- Banning overtime
- Changing deadlines and teams
- Changing or crashing the hardware (!)
- Disrupting file stores

(Dawson 2000)

When discussing the results of this experiment, (Dawson 2000, p. 217) states that "The real evidence (...) is in employers' reaction to graduates (...). Employers have reported back to the university that they found the students (...) to be particularly well prepared for the workplace".

Hence, one way of mitigating task-related issues with SE is to improve the training of software engineers and developers. It is important to note that process models can indeed help to address some of the prevailing difficulties even if they fail to consider the described eventualities: for example, the concept of RUP (see 2.3.5) can be helpful in diminishing risk brought about by frequently changing requirements. The role of process models to solve SE issues is discussed in more detail in subsection 2.4.3.
Dawson (2000, p. 218) also concludes that "If, as many employers believe, computer science and software engineering graduates are 'very knowledgeable, but not a lot of use' then there is much to be gained by playing a few 'dirty tricks' both at university and during company induction programmes to introduce them to the realities of a real software engineering development".
The latter statement however has to be handled with care, because it raises questions concerning the purpose of academic degrees: for example, should universities place an emphasis on conveying theoretical knowledge or should scholarly teaching rather be focused on preparing people for the working environment? Parnas (1999) sheds some light on this issue by arguing that practical software engineering courses at universities can- and even should coexist with more theoretical computer science programs. (Section 2.5 provides a more in-depth discussion of the academic situation surrounding the discipline of SE).

Difficulties with individual SE tasks other than the issues described above have been mitigated through technological advances over the past decades. Integrated development environments (IDEs) helped to increase the productivity by providing useful features, as e.g. automated error highlighting in source code as well as IntelliSense[8]. Furthermore, programming libraries such as the Standard Template Library (STL) in C++ contain a collection of algorithms and data structures, which can reduce development expenses through mitigation of the overall workload for the programmer. Also, this type of reuse has the potential to improve the quality of the code as it decreases the likelihood that developers introduce errors in e.g. complex standard algorithms. In addition, high-level languages aid in abating the complexity of programming languages (Brooks 1987).

Results of a study on *runaway*[9] projects, described in KPMG (1995 cited in Glass 1998, p. 15) show that despite the reduction of the difficulties described above, "Technology is a rapidly increasing *cause* of runaway projects".
One possible reason for this may be that technological advances could reduce the complexity of expressing and writing the program, but they might not reduce the complexity of the software entity itself. Northrop (in Fraser et al 2007, p. 1028) states that "Innovations have too often focused on the accidents not the essence and in some cases have added greater complexity to software production". Complexity and essence inherent in the nature of software are discussed in more detail in the following subsection.

[8] *IntelliSense* is the name for a feature in the Microsoft IDE *Visual Studio* and allows to automated completion of instructions while writing the code. This feature is especially valuable when using standard libraries.

[9] A *runaway* project "(...) is a [software] project that goes out of control primarily because of the difficulty of building the software needed by the system" (Glass 1998, p. 3).

2.4.2 The Nature of Software

Some of the problems explained above resemble "accidental difficulties" from the infamous article *No Silver Bullet*[10] by Frederick P. Brooks Jr. (Brooks 1987): accidental difficulties are issues encountered when creating software, but which are not inherent in the nature of software itself. For example, low-level machine languages are prone to attract syntactic errors. Albeit these accidental difficulties can be mitigated through technological progress, Brooks (1987) identifies *conformity, changeability, invisibility* and *complexity* as essential difficulties of software, which may be much harder to come by than through improvement in technology.

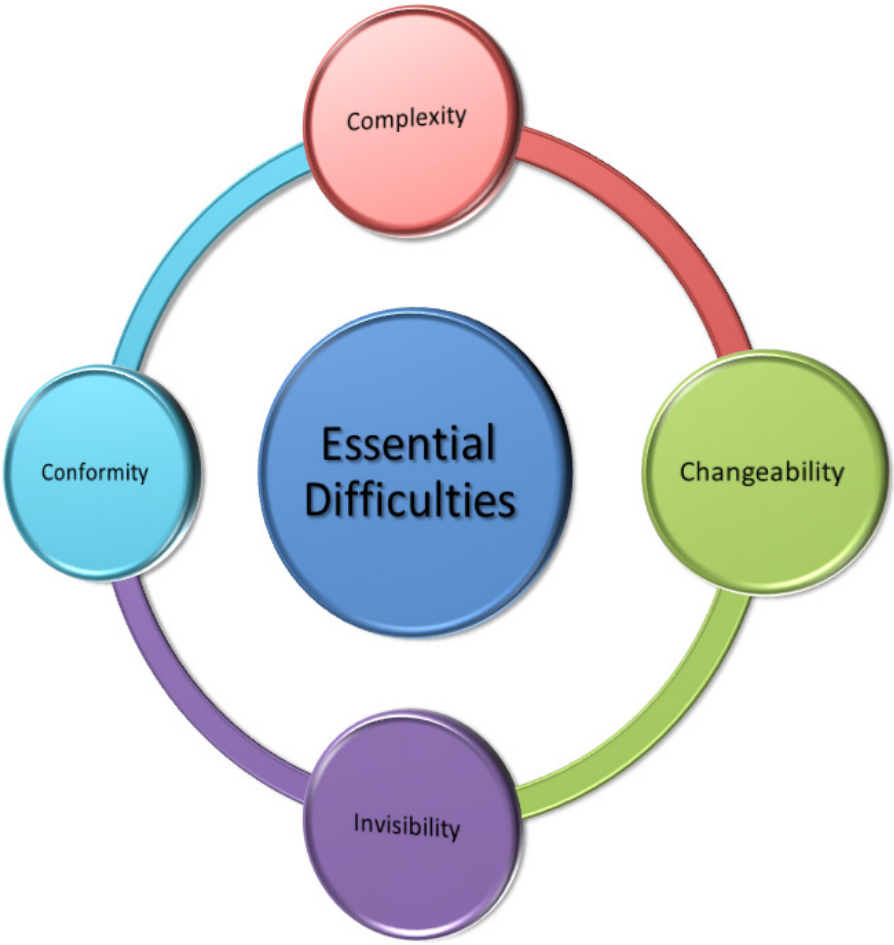

Figure 8 – Essential difficulties of software as identified by F. Brooks (1987)

Conformity refers to the interfaces which software components provide in order to communicate with other components or applications. These interfaces in turn contribute vastly to complexity which "(...) cannot be simplified out by any redesign of the

[10] Brooks (1995) makes use of the term *silver bullet* as an allegory for the ultimate solution to difficulties with software projects, which sometimes have the tendency to become uncontrollable *werewolf* monsters.

software alone" (Brooks 1987, p. 12). **Changeability** denotes the changing environment in which software operates. This includes neighbouring applications, users, underlying hardware and even laws. "These all change continually, and their changes inexorably force change upon the software product" (Brooks 1987, p. 12).

With regard to **complexity**, Brooks (1987) argues that software systems can have orders of magnitude more states than digital computers. Furthermore, function- and method invocation is a concept that can increase the complexity of process flows in a non-linear fashion. **Invisibility** refers to software being difficult to visualise, because in many cases, more than one type of diagram may be needed in order to depict structure, functionality and data flow of an application. Hence, despite even complex constructs such as computer chips can be visualised, there is no similar geometric abstraction possibility for software (Brooks 1987). The usage of the Unified Modeling Language (UML) to create e.g. class diagrams, use case diagrams and process flows can help to mitigate this issue. However, "The reality of software is not inherently embedded in space" (Brooks 1987, p. 12). Therefore, software is less "visible" and thus more elusive than most products.

The author is of the opinion that out of the four difficulties presented above, complexity and (resulting) invisibility may be the main obstacles one has to face when applying engineering principles from other disciplines as e.g. construction engineering to the creation of software.

For example, science based on math and physics often uses simplified models of complex phenomena. A strong argument why this might not simply be applied to software has been made by Brooks (1987, p. 11): "This [simplified models] worked because the complexities ignored in the models were not the essential properties of the phenomena. It does not work when the complexities are the essence".

Glass (1998, p. 4) ascribes inaccurate cost estimations for software projects to the complexity of constructing software and states that "(...) some say it is the most complex task ever undertaken by human beings".

> *"I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors (...) but they are fuzz compared to the conceptual errors in most systems.*
> ***If this is true, building software will always be hard. There is inherently no silver bullet."***
>
> (Brooks 1987, p. 11)

Lopez however argues in Fraser et al (2007, p. 1028) that "There is a 'silver bullet' – it is the pursuit of personal and professional excellence – this when achieved, easily gives us an order of magnitude improvement in software productivity. There is no 'silver

bullet' from without – it must come from within".
Northrop (in Fraser et al 2007) claims not to have come across a silver bullet and raises the demand for more "great" software designers, too.

Parnas (cited in Fraser et al 2007, p. 1029) disagrees with this view and explains that there is much routine work which needs to be done properly: "If we were relying on such people [designers, architects and programmers of 'great insight and creativity'] to build our bridges, the ferry industry would be in great shape". Sharing the view of Brooks (1987) regarding complexity, he also states that progress in SE can stem from building increasingly simple systems.

Apart from the argument that software should be developed incrementally and therefore be "grown" as mentioned in 2.3.4, Brooks (1995) also states that prototyping and focus on requirements refinement can be promising ideas to mitigate difficulties with the conceptual essence. In addition, he describes software reuse as such a respective option. Sommerville (2007) considers software reuse as a means of reducing cost, risk and duration of software projects as well.

2.4.3 SE Models and Methods

Perhaps the most obvious problem with standard SE methods and models may be that they often fail to address the problematic described in the preceding subsections. One could argue that it might not be the fault of the particular process model or SE method if not every possible disruption scenario for projects is covered: after all, the purpose of the software model can be at best to outline means of development.
However, this assumption constitutes a major problem, because planning for software projects often heavily depends on models – especially with inexperienced teams or project managers. Boehm (1988) expresses difficulties with his spiral model, stating that the reliance upon peoples' expertise when it comes to risk assessment is one of the hurdles to broad successful usage of the spiral model. If the reality of the project execution accumulates contingencies which are not considered in the model and during planning stages, then the project can have a relatively high likelihood to deviate from the plan. Hence, there is an apparent need for SE models to address more of the difficulties which could be encountered during SE projects.

As mentioned in 2.4.1, some models already mitigate problems stemming from individual tasks. But can process models and methods consider all major contingencies and eventualities without losing flexibility and without becoming "too heavy" for efficient software engineering?

Brooks (1987, p. 17) presents his view concerning the planning stages of software development: "Much of present-day software acquisition procedures rests *[sic]* upon the assumption that one can specify a satisfactory system in advance (...). I think this assumption is fundamentally wrong, and that many software acquisition problems spring from that fallacy".

In conclusion, the issues presented in this section seem to require solutions on a level which is different from SE models and methods. In order to conduct respective further investigation, the following section explores scholarly attempts to solve the difficulties with SE.

2.5 Academic Solutions or further Practical Problems?

2.5.1 Body of Knowledge & Education

Parnas (1999, p. 20) states that "Because of a rigid accreditation process (...) there is a well-documented 'core body of knowledge' for each of the established engineering disciplines. There is no corresponding body of knowledge for computer science". The Software Engineering Body of Knowledge (SWEBOK) addresses this issue by providing a compiled guide to widely acknowledged literature in the field: "For software engineering to be fully known as a legitimate engineering discipline and a recognized profession, consensus on a core body of knowledge is imperative" (Abran et al 2004, p. 1-1).

With regard to education, subsection 2.4.1 introduces the idea of a coexistence of Computer Science (CS) programs (i.e. courses) and SE programs at universities: "Just as the scientific basis of electrical engineering is primarily physics, the scientific basis of software engineering is primarily computer science" (Parnas 1999, p. 20). He continues to state that "In the SE program, the priority will be usefulness and applicability; for the CS program it is important to give priority to intellectual interest (...)" (Parnas 1999, p. 26).

2.5.2 Accreditation as a potential Silver Bullet

Concerning accreditation of academic SE programs, the SWEBOK-guide supports the following view: "Without such a consensus [on a body of knowledge], no licensing examination can be validated, no curriculum can prepare an individual for an examination, and no criteria can be formulated for accrediting a curriculum" (Abran et al 2004, p. 1-1). Accreditation and consensus on a body of knowledge can facilitate customer confidence in software organisations and is vital to legislative acknowledgement as shown in 2.6.

Scientific SE projects seem to be less likely to reflect the described problems as opposed to commercial software projects: a recent example for a vast but successful scientific undertaking can be found in the storage and data management for the Large Hadron Collider (LHC) experiment at CERN: "(...) the very difficulties involved in providing data management at such a scale have also ensured that the development of genuinely robust and performant *[sic]* grid level middleware is well on its way to being completed" (Stewart et al 2007, p. 77).
Furthermore, internal research- and development (R&D) software projects which can be encountered mainly in large organisations, seem to cultivate similar positive condi-

tions because they "(...) have a great deal of flexibility and freedom to accommodate (...) such practices as prototyping, evolutionary development, or design-to-cost" (Boehm 1988, p. 70).

Therefore, accreditation of SE courses is likely to improve the situation for projects predominantly focusing on

- Software development for scientific endeavours
- Internal corporate R&D undertakings

It becomes apparent that the range of projects which may benefit from accreditation is by far not wide enough.

> *Software engineering will not achieve the status of a true profession until it has a similar accreditation system. (...) They [software engineers] are expected to be able to apply mathematics and science (...) to assure that the system they design will perform its tasks properly when delivered to a customer."*
>
> (Parnas 1999, p. 23)

The argument of Parnas above implies that the nature of all projects which software engineers work on is related to engineering. Software engineers however work on non-engineering related software projects, too. Therefore it could be a big mistake to interpret the statement of Parnas (1999) above in such a way which might form the assumption that a unified SE body of knowledge and accreditation can magically solve all of the problems which the industry faces to date and will have to face in the future. The example provided in this subsection is to emphasise the favourable conditions of certain scientific- and internal R&D projects over commercial ones, which on the contrary can often be driven by high pressure resulting from competition and strategic intentions rather than innovation or the achievement of high quality.

Hence, accreditation can help to improve the situation with SE in certain areas. However, the author is convinced that accreditation may not solve the difficulties SE is facing predominantly in competitive commercial areas such as Business Intelligence (BI). With regard to consensus on a body of knowledge, the author believes that over the long term, an increasingly unified knowledge base might bring about a more positive situation in all areas of SE. This may include scientific undertakings, advocated large-scale internal developments as well as commercial software projects.

2.6 Legal Implications

Speed (1999) describes the legal aspects of accreditation using the example of the American state Texas:

> *It is illegal to practice software engineering in Texas without a license. (...) [The] development of software for engineered systems — including embedded systems, real-time systems (...) — is software engineering. A person who performs software development work in these areas without a professional engineering license or exemption is breaking the law.*
>
> (Speed 1999, p. 48)

Bott (2001, p. 16) explains the situation in the U.S. as follows: "(...) the only people who are formally allowed to describe themselves as software engineers are those who are licensed in some other branch of the discipline and a company can only describe itself as a software engineering company if it employs at least one professional engineer licensed in another branch". He also cites the so-called "Finniston Report" which states, that "In every overseas country at which we looked the status of engineers and engineering was high; it attracted high quality entrants and was accorded a priority in social and industrial affairs that is generally lacking in the UK" (HMSO 1980 in Bott 2001, p. 7). In comparing respective regulations of the U.S. with the UK, he explains the situation as follows:

> *The engineering profession is much more highly regarded in the USA than in the United Kingdom and is subject to strict statutory legislation, going further than what the Finniston Report recommended. The title of engineer is protected and the practice of engineering is restricted. Very similar legislation exists in Canada.*
>
> (Bott 2001, p. 14)

Jones (1995, p. 99) warns that "If the software engineering community cannot rise to the level of becoming a recognized profession and engineering discipline, we face an uncertain future with ever-mounting prospects of unfriendly legislation and harmful government actions".
Therefore, even though accreditation may not solve the difficulties with SE in commercial areas as concluded in the previous section, nevertheless it may provide a solution to legislative issues commercial software companies might soon have to face.

2.7 The History of Engineering and Medicine in Contrast with SE

In the 18th century, formal groups of engineers began to emerge; the first being the French Group for Bridges and Roads (Corps des Ponts et Chaussées), founded in 1716 in Paris (Bott 2001). "(...) a society of civil engineers was formed in England later in the century. It was also in Britain that the first engineering grouping that aimed to represent the profession, and thus to be a professional body in the modern sense, was formed" (Bott 2001, p. 6). The British Computer Society (BCS) which was founded in 1957, started with the purpose to exchange and disseminate knowledge about computing (Bott 2001).

"Large civil engineering structures have been built since before recorded history but it is only in the last 150 years or so that their design and construction has been based on theoretical understanding rather than intuition and accumulated experience" (Grimson & Kugler 2000, p. 543). The authors cite Maginnis (2000) to emphasise the "(...) need to move towards a certification process for software engineers" (in Grimson & Kugler 2000, p. 544).

Jones (1995) raises the question of what makes an engineering profession. He provides a list of factors associated with recognised engineering such as a well-defined body of knowledge, qualifying examinations and a code of ethical responsibilities; to name only a few. An interesting aspect of his paper is the comparison of the software community with the medical profession:
"It's been only 150 years or so since medical practice faced challenges similar to those software faces today: It was an amorphous and fragmented community with many questionable and unproven practices, and academic training spanned every possibility from state of the art to totally inept" (Jones 1995, p. 99).

Lord Goodman (1986 cited in Grimson & Kugler 2000) criticises computer science education by stating that people who were trained solely in this profession are uneducated persons. Grimson & Kugler (2000, p. 543) formulate their viewpoint as follows: "It is interesting to recall that the very same criticisms were levelled against universities which sought to introduce engineering degrees (...) in the mid-19th century".

2.8 The Potential Origin of Intrinsic Difficulties with SE

2.8.1 Terminology

The IEEE Standard Glossary of Software Engineering Terminology defines SE as follows: "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" (in Abran et al 2004, p.1-1). Furthermore, the definition includes research and educational aspects by adding "(2) The study of approaches as in (1)".

Sommerville (1993, p.7) states "Software Engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use". Albeit Sommerville recognises the systematic approach of software engineers in order to produce high-quality deliverables, he also concedes that the development of software is different from approaches in other engineering disciplines: "Less formal development is particularly appropriate for the development of web-based systems (...)" (Sommerville 1993, p.7).

Boehm combines definitions of the terms software and engineering in the following manner:

> *Software engineering is the application of science and mathematics by which the capabilities of computer equipment are made useful to man via computer programs, procedures, and associated documentation.*
>
> (Boehm 1981, p. 16)

With regard to the NATO conference in 1968 where the term was coined (see 2.2), Dijkstra (1993) "(...) interpreted the introduction of the term 'software engineering' as an apt reflection of the fact that the design of software systems was an activity par excellence for the mathematical engineer. **But that interpretation of the term did not sail well. (...) software engineering has become an almost empty term (...)".**

When considering the difficulties with SE presented in this chapter, an intriguing question must be raised: does the term *Software Engineering* capture the nature of this discipline with sufficient accuracy? This question could be of high relevance; especially for the usage of some process models and methods that may have potentially been inspired by- or derived from other engineering disciplines.

One cannot be sure whether a different choice of words would have created less severe issues for the discipline of SE as a whole. Bryant (2000, p. 78) however argues that "(...) the term itself carries with it a set of mixed cognitive implications that contribute to the intellectual quandary surrounding software development". In other words, the term itself may often have caused people to approach software development in an unsuitable manner.

According to Bryant (2000, p. 78), this dilemma manifests itself in the fact that software developers "(...) seem to wish to mimic engineers, and lay a claim for the status of an engineering discipline; **but commercial demands and consumer desires do not support this".**

Even the NATO Conference Report, which essentially attests to the creation of the term software engineering, contains an intriguing statement:

> *The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.*
>
> (Naur & Randell 1969, p. 8)

Given that an engineering approach may not be suitable for most commercial software projects, creating the term software engineering out of provocative stimulus appears to not have been a wise choice. However to be fair, it was virtually impossible for attendees to the conference in 1968 to foresee the evolvement of the discipline over the decades that lay ahead.

A further indication of how delicate this terminological issue seems to be is contained in the introduction to the paper "A View of 20th and 21st Century Software Engineering". In this relatively recent paper, Boehm (2006) cites Merriam-Webster (2000) in order to provide a definition for the word *engineering.* If the terminology alone would not be an issue, then a contemporary scholarly paper on software engineering would not need to make statements in its introduction to define the discipline it is aimed at. Hence, the ambiguousness of the term *software engineering* seems to have not been mitigated despite advances in the field of computing.

2.8.2 Managerial Aspects

An experiment was conducted with students who used an application which is a simulator for software projects. Drappa and Ludewig (2000, p. 199) describe the motivation behind the experiment as follows: "The best (and probably the only) motivation for learning software project management is experience from projects that failed due to insufficient management". Drappa and Ludewig (2000, p. 207) come to a sobering conclusion: software engineering "(...) is a strange field because its message is extremely simple, but still too demanding for most people. They pretend they understand, but their behavior shows that they do not". Albeit this statement was made with regard to students rather than experienced experts, the findings should be taken serious for the following reason:
The analysis of the simulated project experiment with students shows that students repeatedly made the same mistakes again. They did not reflect upon the outcomes of their simulated projects (Drappa & Ludewig 2000). Most graduates will probably be-

come more experienced as they progress with their careers in IT, but the question remains whether the majority of specialists will change the inappropriate behaviour pattern described above during their working-lives. Therefore, improving the situation concerning project management competencies of students is important, given that graduates with high potential are likely to become managers. If potential future leaders are not willing to learn how to conduct SE the right way, they are prone to repeat the mistakes which were made in the past over and over again. This in turn can lead the issues with IT projects to prevail even longer into the future. Hence, education in the field of SE has to address this problem with more intensity.

"Study after study of software estimation in practice has shown that most often cost and schedule targets are set by marketers or customers, next most often by managers[11], and least often by the technologists who will do the work. Without control of their own destiny, practitioners still get blamed when things go wrong" (Glass 1998, p.8). Even though Glass (1998) ascribes deficient estimations to the choice of personnel, he concedes how little is known in general about accurate schedule estimation for software projects. "(...) it is simply the norm that schedule targets, and thus cost targets, are unreasonably short. Thus project achievement of them is at best problematic" (Glass 1998, p. 11). van Genuchten (1991, p. 589) combines findings from previous surveys with his own studies, concluding that "Over-optimistic planning was cited as a probable cause in all the studies which examined reasons for delay". Concerning managerial aspects, the most intriguing finding is as follows: project leaders and general managers disagree on 60% of the reasons why projects ended up with schedule- and budget overruns (van Genuchten 1991).

Grimson & Kugler (2000, p. 542) found that none of the causes of software systems failures identified by Glass (1998) are technological: "(...) they are all related to software management". Drappa & Ludewig (2000) share this view and state that the development of large software systems in particular constitutes a non-technical problem. They justify this statement by describing the importance of planning and sound analysis.

However, one needs to raise the question whether the pressures resulting from managerial – and financial for that matter – demands are unique to the software industry? Glass (1998, p. 11) presents his view concerning other knowledge fields: "Given the intense competitive pressures of the last half of the 20th century, workers in all fiends find themselves under the gun to do more than they can in too short a time". Despite having stated this argument, he continues to concede that "The problem for systems

[11] With regard to cost and schedule estimations conducted by managers, it needs to be emphasised that not all managers are technically inexperienced. Many people working in leading positions may have gained experience in technical roles prior to their promotion to management levels.

and software is worsened, however, by the fact that the field is so young, and because we know so little about it compared to other fields, we are really never quite sure that an unreasonable schedule is, in fact, THAT unreasonable" (Glass 1998, p.11).

2.8.3 Emotional Factors

Glass (1998) denies the existence of a "software crisis", arguing that computers and software are not only dependable and indispensable, but have proven successful in many fields such as e.g. the provision of plane reservations, banking transactions and by serving as equipment for space exploration. He blames vendors, researchers and academics for "crying crisis" because in his view, they all gain something from keeping the picture of a crisis alive. The findings of this chapter support some of the statements made by Glass (1998): not all areas within the discipline are equally prone to the difficulties encountered with software engineering. However, his view to deny the existence of a crisis oversimplifies what different people may mean when they make use of the word "crisis". The utterly normal maturing process of a new discipline has sometimes been denoted as a crisis, because difficulties with SE have caused and still continue to cause many commercial projects to be delayed or even to fail.

Some statements by Glass (1998) are of emotional nature: he expresses what could be regarded as heroic solidarity with- and appreciation for "(...) all those pioneering software practitioners who have struggled through the barriers in their path to make our era truly the Computing era (...) May they be appreciated for their efforts, and lauded for their success" (Glass 1998, p. 8).
The irony contained in the statement above is as follows: on the same page, Glass (1998) asserts that estimation of software projects often does not lie in the hands of practising experts (see citation in section 2.8.2). This however implies the notion that software developers should get appreciated if a project turns out to be successful, but that they cannot be held responsible in case a project fails (!) In his conclusion, Glass (1998) addresses himself and fellow developers (who are presumably the intended readership) as "nerds". Affirmation of stereotypes could be deemed unprofessional and does not help the discipline at all, because the illusory separation between IT specialists and other business departments may in fact be the biggest issue faced by software engineering to date.

On the one hand, Glass (1998) has written a great book in which he identifies the important point that practitioners and experts are seemingly ignored when it comes to the estimation of software projects. On the other hand, he fuels the dubious conflict among internal corporate departments by strengthening the image of the typical "nerd" who might be treated unfairly by managers. A similar example in the field of engineering is provided by Bryant (2000, p. 85). He cites Johnson (1972) to argue that "(...) far from being natural decision-makers, engineers were in fact imbued with a 'trained incapacity for thinking about and dealing with human affairs'". Referring to the article *No Silver Bullet* written by Brooks (1987), Fowler (in Fraser et al 2007, p. 1028) has the follow-

ing view about this issue: “Sadly many big purchasing decisions in the corporate world are not made by technologists – these days the werewolves are on the golf courses”.

Experts have to get over this immature behaviour of making such statements which could lead to things only getting worse. The job of software experts is to think ahead and work out how the situation can be improved *from a software engineering point of view* without pointing fingers at one another.

2.8.4 Future Outlook

Glass (1994) predicts a vision of the year 2020 where both, the SE research community and practitioners, work closer and more efficient together. Better collaboration can indeed be helpful to the discipline, as long as fairness is guaranteed: business factors of SE projects such as corporate profit-driven interests (esp. in the short term) may not always be reconcilable with research interests – even if this concerns internal research.

2.9 Summary and Conclusion

The preceding sections form a literature review on software engineering. The chapter covers the emergence of the discipline during the "early days" of software development, introduces major SE concepts and discusses difficulties faced by the discipline. In addition to contrasting potential solutions for these difficulties, the review explores academic and legal aspects. Investigation into potential problem sources comprises the analysis of terminological, managerial and emotional aspects including their impact on SE.

2.9.1 The Measures to Improve the Situation with Software Engineering

The literature review has elicited measures which are discussed by scholars as promising solutions to prevailing difficulties with SE:

1. Adaption of proven engineering principles for software engineering
2. Engineering-focused education of software engineers
3. Accreditation of the discipline
4. Consensus on a body of knowledge
5. Agile methods

2.9.1.1 What the measures can achieve

These measures might be able to mitigate essential difficulties of software described in 2.4.2 such as conformity (e.g. by adapting engineering principles for sound definition of interfaces) and changeability (e.g. by using agile development for requirements elicitation, which can mitigate negative effects of frequently changing user requirements). Consensus on a body of knowledge might turn out to be equally helpful because the first step to controlling complexity and improving visibility of software architectures is to have unified ways in which SE experts do their work. Furthermore, some agile methods have an incremental nature which grows the software gradually and thus can help developers to stay more in charge of increasing complexity – at least during the development phases (see the adequate statement by Brooks in section 2.3.4 about software being grown rather than being built).

Also, the literature review has found that the measures 1–4 could potentially improve the quality of those SE projects, which are

- Engineering-related (e.g. real-time systems)
- Scientific & non-profit
- Internal R&D
- Safety critical.

For example, engineering-related software projects can benefit from accreditation, because it forms an important basis for legislation (see 2.6): the legal environment could define whether "pure" software companies are permitted to conduct engineering-related software projects.
Both scientific and non-profit-driven projects have no typical marketplace competition as their purpose does not accompany a commercial business goal but rather serves the purpose of the research.

Therefore, this type of projects could make better use of document-driven approaches than other types of software developments. The reason might be that these projects are less likely to run the risk of frequent changes in major requirements: end-users are often experts, specialists and engineers who may have a clear view on the exact purpose of the software as opposed to requirements elicitation from corporate business users, clerks and office staff. The latter may even be reluctant to accepting the potential change brought about by the new software system, which makes collection of requirements less precise and more difficult.
Similar favourable conditions for the usage of document-driven approaches as is the case with scientific projects, apply to internal R&D undertakings; with the only difference that profit is more likely to become a goal in the long run. With regard to safety-critical software, these systems tend to have high quality requirements (e.g. zero error rate), which in turn can trigger the need for detailed engineering-like documentation and test.

2.9.1.2 What the measures cannot achieve

Despite the beneficial effects of the measures described above, the author is convinced that these measures will not be able to entirely solve the "essential difficulties" of complexity and invisibility. The problematic of complexity being *inherent* in software is described in 2.4.2.
Concerning invisibility, the measures may pose solutions that are not practical enough: a better way to mitigate this issue could be improvement of existing representation techniques for software architecture (e.g. 3D visualisation).

A significant problem with the measures presented above is that they are not equally effective on all types of software. Hence the sole and imprudent application of these measures to the entire software engineering discipline could even cause delays in solving prevailing issues. This problematic is described in more detail in the following subsection:

2.9.2 The Dilemma: Contradicting Methods

The scholarly literature of software engineering reveals a contradiction: on the one hand, some authors claim a demand for adaption of engineering principles to SE. On the other hand, agile methods that work in the exact opposite direction are being advocated. Unfortunately, sources demanding either solution do not consistently make clear which type of software project the respective measure should be aimed at.

It is obvious that measure number five (agile methods) seems to contradict the measures number one and two (an engineering-like approach). The reason why calls for implementation of measures 1, 2 and 5 coexist despite this apparent contradiction, might be as follows: diversified measures may have evolved naturally and implicitly as different types of software projects require different types of approaches. In the search for a silver bullet however, many sources may have failed to investigate general suitability of the respective measure.

Commercial profit-driven business application development in a competitive corporate environment could be the type of software which is less likely to profit from implementation of measures 1–4. Despite this assumption, the author is of the opinion that measures number 3 (accreditation) and 4 (body of knowledge) can indeed improve customer confidence and facilitate cultivation of sound practices. However, these measures may be too unspecific to prevent commercial SE project failures.

The reason why engineering-related measures might not mitigate prevailing issues with some commercial projects could be that these projects possess attributes for which an engineering approach may not be an ideal choice. As mentioned above, different types of software products require different types of development approaches. The emergence of iterative-, incremental- and agile methods (see 2.3.5 and 2.3.6) indicate a general need for more appropriate and innovative development approaches to e.g. BI systems, non-safety critical office software as well as web-applications.

2.9.3 The Research Question: Implicit Corporate Business Factors

As elucidated in this chapter, iterative and incremental approaches can be helpful when a project faces frequently changing requirements. Despite these improvements on the level of the SE models and methods, many commercial IT ventures continue to fail as highlighted in 2.4. Thus, the question raised in 2.4.3 remains important: can process models and methods consider all major contingencies and eventualities without losing flexibility and without becoming “too heavy” for efficient software engineering?

Since even optimised models and methods turned out to be helpful to only a certain extent, evidence from this literature review leads to believe that the source of problems with commercial software applications developed in a competitive and profit-driven environment may not solely originate from the actual approach. For example, managerial factors (see 2.8.2) were identified to have a vast impact on software projects.

Therefore, more sophisticated process models and methods alone may not solve prevailing difficulties of commercial software projects. This leads to the question posed in the introduction chapter:

- Is recognition and consideration of implicit corporate business factors and integration of these factors into planning for commercial SE projects a potential key to successful software engineering projects?

2.9.4 Concluding Thoughts

Qualitative improvement of commercial SE projects requires an investigation into external factors to the project. These factors comprise e.g. managerial aspects and corporate goals. The following chapters present research findings on this topic and describe the development of a practical solution which is expected to improve the situation for commercial SE projects.

Chapter Three

The preceding chapter consolidates the term software engineering. Part of the literature review conducts investigation into prevailing difficulties with a specific type of SE venture. This undertaking was identified in chapter two as commercial software projects (usually in a competitive profit-driven environment). Also, findings of the previous chapter indicate that managerial aspects can have a major impact on the successful outcome of commercial SE projects.

3 Planning for Successful Commercial Software Engineering Projects

3.1 Introduction

This chapter outlines the characteristics of commercial SE and defines what may constitute a successful project of this kind. In order to link abstract SE methods and models to practical applications, section 3.4 provides an overview of project planning tools and analyses the suitability of these applications (3.5) for SE projects. Investigation into general project planning and management concepts is conducted in section 3.6, before the chapter closes with a conclusion concerning the potentials as well as the difficulties of using these concepts and tools for SE.
Albeit the chapter starts out with a focus on commercial SE projects, most of the sections address general software engineering. However, commercial SE projects are described in detail in this chapter as this forms the basis for further research in chapters four and five.

3.2 Exemplifying Commercial SE Projects

3.2.1 Business Application Development

Business applications constitute non-safety-critical[12] business software. One of its main purposes is to maximise efficiency and reduce cost across corporate departments such as e.g. human resources, marketing and accounting.

Some examples for business applications are

- e-Business (Electronic Business)
- Enterprise Resource Planning (ERP)
    - Supply Chain Management (SCM) – e.g. details of suppliers
    - Customer Relationship Management (CRM)
    - Product Lifecycle Management (PLM)
    - Business Intelligence and Data Warehousing
    - Software for accounting purposes
- Software to support product development: Computer Aided Design (CAD)

Some ERP and CAD systems can be purchased as ready-to-use software. Adapting such applications to a particular client organisation may require modifications which in turn can result in higher costs.

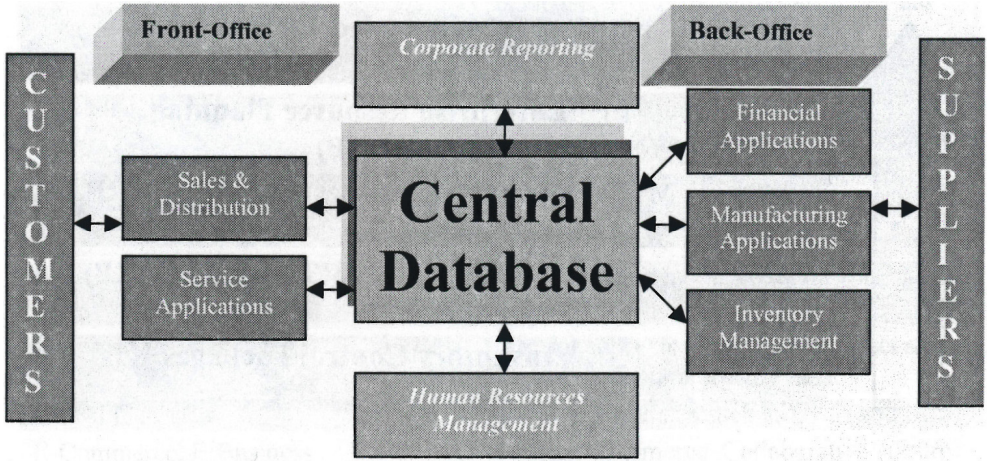

Figure 9 – ERP systems concept (Nah 2002, p. 37)

Furthermore, the following type of business software can be of great interest with regard to analysis of commercial SE projects:

[12] *Non-safety-critical* refers to the safety of human beings. Hence, failure of the system poses no ultimate threat to human lives.

- Bespoke individual software applications that are knowledge-specific computer programs for a certain field, as for example
    - Artificial Intelligence (AI) systems to support consulting activities
    - 3D computer graphics engines
    - Electronic Commerce (EC) software (e.g. corporate website)

Bespoke SE projects may be relevant to the research described in this work because usually no blueprint exists for the software which is to be developed. Specialised applications for certain business areas can be vital in enabling or maintaining the strategic advantage of an organisation over its competitors as shown in 4.3.

3.2.2 Commercial Real-Time Application Development

"The notion of real-time system, as it is currently accepted, describes a computational process that has to respond to internal or external stimuli in determined periods of time" (Ionescu & Cornell 2007, p. 3). Examples for commercial real-time applications are radar-systems in aircrafts, software for the management of nuclear power plants and software to control airbags in cars.
Commercial real-time applications are taken into account throughout the research, for this type of software project has an intriguing potential: it seems to be able to mitigate some negative effects of the factors such as managerial intervention (see 4.4.2).

3.3 What Constitutes a Successful Commercial SE Project?

Depending on which stakeholder is concerned, the views of what constitutes success in SE projects can vary strongly. For example, customers may require the software to have as few bugs (i.e. errors) as possible; preferably none. High quality is expensive and thus sometimes it might be difficult for management of software companies to find the balance between customer satisfaction and internal development costs. Managers of software companies might deem projects successful which are delivered within a reasonable time span, functionality which might have some bugs but allows the customer to work with (e.g. core modules) and in some cases a seamless transition to maintenance contracts after which bug-fixing can be commenced. Hence, management may be of the opinion that a successful SE project is one which may not be perfect from a technical point of view, but which satisfies the majority of stakeholders and generates profit for the software organisation.

Therefore, taking into account the findings from the literature review and the practical experience of the author, a successful commercial SE project possesses the following attributes:

1. The project is delivered with no overrun in budget or time. Given the findings in section 2.4 which are drawn from industry figures, a realistic demand for

schedule overruns would constitute no more than 15-20% of the given project time. However, the goal for SE must be to meet budget and time constraints.

2. The stakeholders are satisfied to an extent where the project can be signed off as delivered. Signing off the contract can be seen as a confirmation that stakeholders are satisfied with the deliverables. However, the remaining uncertainty is that stakeholders who are in fact not satisfied could sign off the project anyway in order not to have a project failure on their record.
3. No additional resources are required other than planned prior to commencing the project.
4. The undertaking generated reasonable profit for the software company, which allows the organisation to grow.

Further attributes can be internal success milestones set by the respective enterprise.

3.4 Common Planning Tools to Implement SE Methodologies

The Merriam-Webster Online Dictionary (2009) defines the word *methodology* as follows:

1. "a body of methods, rules, and postulates employed by a discipline: a particular procedure or set of procedures"
2. "the analysis of the principles or procedures of inquiry in a particular field"

Some of the models and methods described in detail in the literature review can be combined to form methodologies. For example, the implementation stage of the waterfall process model may be conducted using XP (see 2.3).

The purpose of this section is to shed light on the efficiency of some common tools, which can help implementing planning methodologies that are often rather abstract. The tools represent only a small fraction of applications which are available in the field of project planning and management. However, the following examples provide an overview of key functionalities and attributes shared among many of these software packages and online services.

3.4.1 Project Planning and Resource Management

This type of software is non-collaborative, which means that it usually stores data in a local project file. The software is not designed for multi-user access e.g. via the intranet of a company. Therefore, only one user can edit a certain project at a time as changes do not get distributed automatically to other users.

3.4.1.1 MS Project

Perhaps one of the most well known products for project planning is Microsoft (MS) Project. Its main functionality enables the user to create a project plan by defining tasks, timelines and milestones. Furthermore, resources can be managed and allocated

to tasks. The software provides different views as e.g. task list, Gantt chart and resource usage. In addition, MS Project is able to display the critical path which emphasises tasks that may be crucial for finishing the project on time.

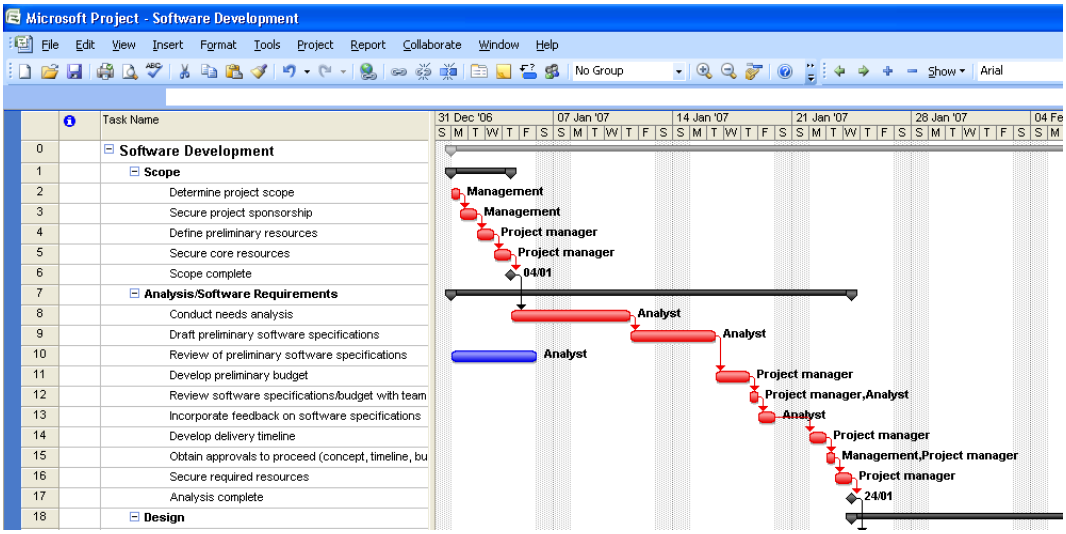

Figure 10 – Partial screenshot of the GUI (Graphical User Interface) of MS Project 2007 displaying a task-list and associated Gantt chart

It requires additional effort from the user of MS Project to allocate resources to individual tasks in a way that prevents both underutilisation and unrealistic working hours. Also, changing project conditions may result in a situation where the user of MS Project has to spend extra time on amending the plan in a consistent manner.

Especially in the field of SME, where companies seem to prefer agile planning approaches, the author has made the experience that companies use predominantly the core functionality of MS Project: for example, the software is used to create tasks, timelines including estimated workload and milestones. This plan can help to provide a visualisation of estimation figures to their customers, which in turn can aid in justification of costs.
However, situations might occur where customers demand actual information on internal resource allocation. Such incidents can be triggered by political motives in case, for example, the customer commissioned other projects with the software company. It could be that the client wants to insure that none of the projects are being neglected due to e.g. reallocation of superior developers to other projects. In this particular case, agile usage of MS Project to simply create an overview of tasks would not be appropriate.

3.4.1.2 OmniPlan

Some project planning software packages, as for example OmniPlan for the platform Mac OS X, resemble the core functionality of MS Project. In addition to slightly different GUIs, these tools may offer some features as e.g. exporting project plans in a suitable format to be imported into calendar applications.

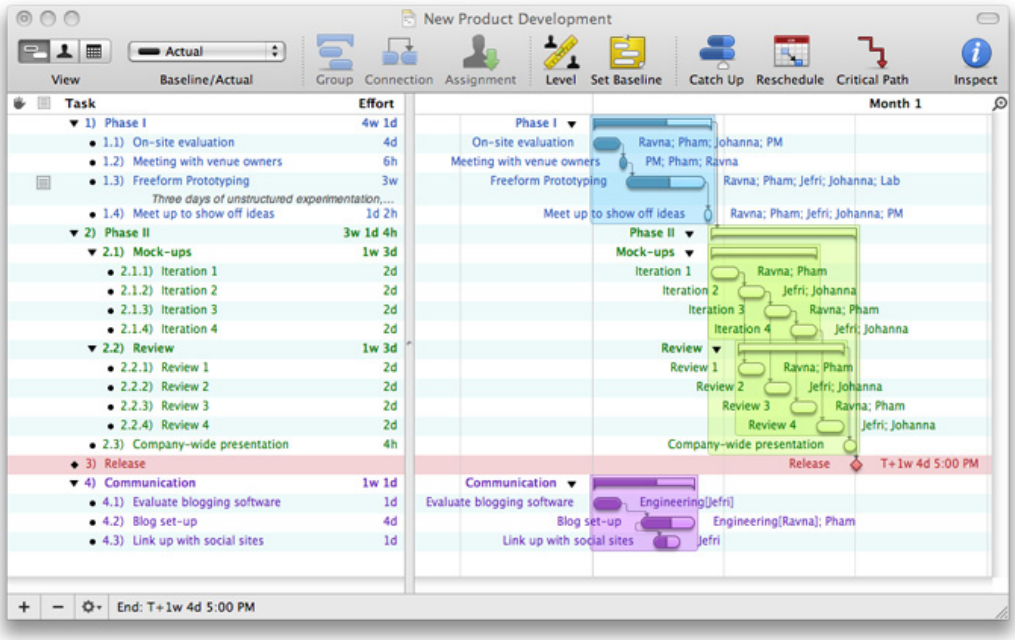

Figure 11 –Screenshot of the software OmniPlan for the platform Mac OS X (The Omni Group 2009)

In conclusion, off-the-shelf software packages like MS Project and OmniPlan can be useful to visualise the workload and to plan for individual tasks. However, extra effort may be required from the user when creating a detailed plan which comprises resource allocation and allows for consistent amendments (e.g. to consider contingencies) as the project progresses.

3.4.2 Collaborative Project Planning and Document Management

Collaborative applications allow an interactive coordination of the project by involving analysts, developers and other stakeholders in the project planning and management process.

3.4.2.1 Zoho Projects

This online project management (PM) platform is hosted online and provides not only collaboration functionality such as forums and document sharing, but also management of tasks, milestones and calendars. In addition, it supports time tracking for accounting purposes and creation of reports containing Gantt-charts (Zoho 2009).

3.4.2.2 Artifact LightHouse

This is perhaps one of the most comprehensible applications which are hosted online. Beside other functions, it encompasses management of changes and tests, requirements management, bug tracking (see 3.4.4), document- and task management as well as general project planning functionality. Furthermore, the application provides timesheets e.g. to measure productivity and profitability of the team (Artifact 2009).

3.4.2.3 MS SharePoint

Some requirements may change during the course of SE projects, which makes centralised document management and versioning indispensable. MS SharePoint provides – beside other features – online collaboration among team members to publish documents, maintain task-lists and create individually customised portals for its users (Microsoft 2007).

3.4.2.4 LiquidPlanner

LiquidPlanner is hosted on Linux servers but supports major web browser applications for client-side usage. Using this software, project plans can be centralised and accessed by individual team members. One particularly intriguing feature of LiquidPlanner is the management of uncertainty in project schedules: the software uses statistics to provide estimations on the likelihood of finishing tasks by certain dates (LiquidPlanner 2008).

3.4.3 Cost Estimation: COCOMO II

Cost estimation for software may be difficult due to the complexity of software (2.4.2). Even though the described project-planning software packages usually include functionality to estimate costs, Boehm et al (2000) provide a more sophisticated model: COCOMO II (COnstructive COst Model).

"COCOMO II is an objective cost model for planning and executing software projects. (…) COCOMO II supports contract negotiations, process improvement analyses, tool purchases, architecture changes, component make/buy tradeoffs, and several other return-on-investment decisions with a credible basis of estimate" (Boehm et al 2000, p. xxvii).

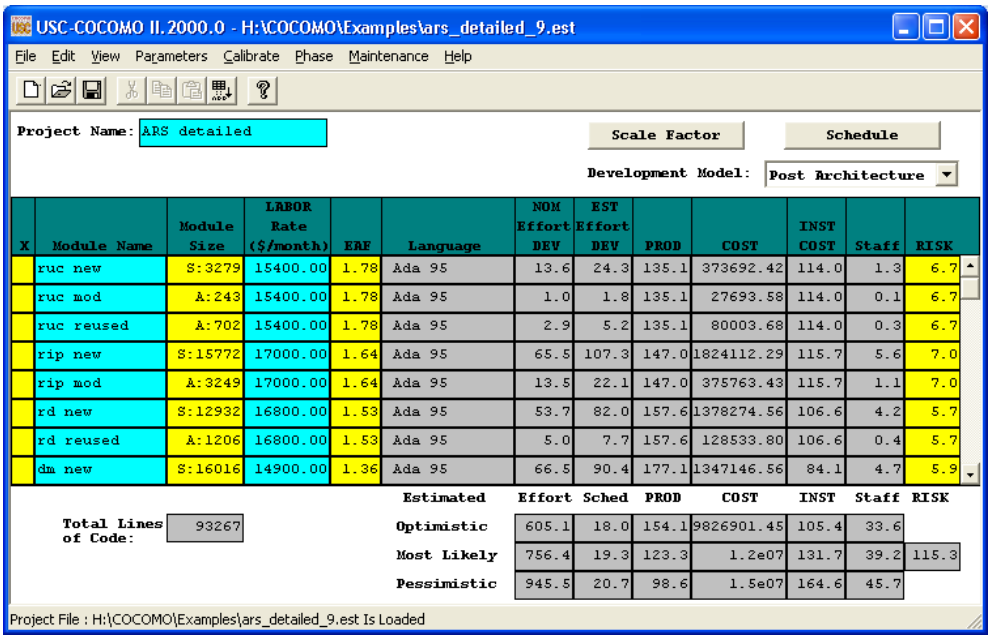

Figure 12 – Screenshot of the cost estimation software COCOMO II

The model provides a range of mathematical equations, but comes with a graphical application which disburdens the user from calculating the formulas manually. However, the software requires the user to understand individual factors such as KASLOC: “Size of the adapted component expressed in thousands of adapted source lines of code” (Boehm 2000, p. 392).

3.4.4 Additional Tools: Project Portfolio Management and Issue Tracking

An example for project portfolio management is the software RationalPlan Multi Project.

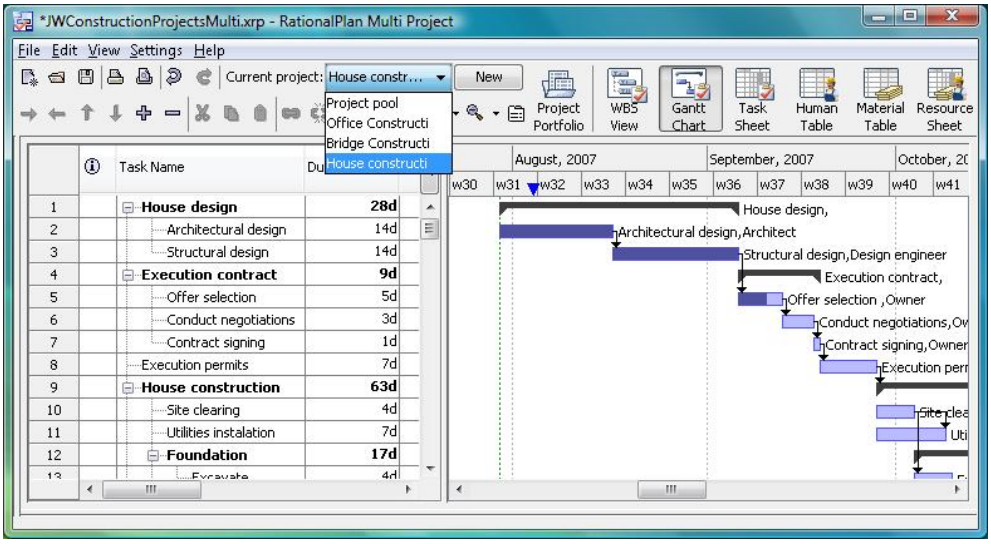

Figure 13 – Screenshot of RationalPlan Multi Project exemplifying the management of a portfolio consisting of several projects (RationalPlan 2009)

This software package enables the user to handle dependencies between projects and to share resources among projects (RationalPlan 2009).

With regard to the management of individual technical tasks, tools like Bugzilla or Trac provide so-called issue tracking (also: “bug-tracking”) functionality, which help to manage and track software errors and individual change requests made by the customer. For example, the project manager or support department can use Trac for task allocation: tickets are created from task information and then assigned to developers.

3.5 Suitability of Planning Tools for Commercial SE Projects

As of the 2nd quarter of the U.S. fiscal year 2009, what do these three people have in common: (1) a software engineer working for NASA (National Aeronautics and Space Administration), (2) a speleologist (cave scientist) and (3) the U.S. secretary of the treasury? – They use Microsoft Project to investigate black holes.

The question needs to be raised why so many software projects still fail to meet deadline- or budget constraints (see section 2.4), despite the availability of allegedly sophisticated planning applications like MS Project. The answer could lie in the generic nature of most planning tools: rather than being specifically aimed at SE project planning, virtually any kind of project which is based on individual tasks can be planned using these rather generic planning applications.

In order to better understand the core of the problem, one has to investigate the nature of a software project and how it differs from projects in other knowledge areas:

3.5.1 Reaping Wheat versus Developing Software

Brooks (1995) makes a statement which can be considered as rather shocking, given that the widely acknowledged de-facto standard for estimating and scheduling effort in software projects is the man-month[13]:

> *"Cost does indeed vary as the product of the number of men and the number of months. Progress does not. Hence the man-month as a unit for measuring the size of a job is a dangerous and deceptive myth.* ***It implies that men and months are interchangeable."***
>
> (Brooks 1995, p. 16)

He argues that men and month can only be interchanged for tasks which require no communication among workers. "This is true of reaping wheat (...) it is not even approximately true of systems programming" (Brooks 1995, p. 16).

[13] *Man-month* refers to the amount of work one person can complete in one month. The same applies to man-hours, man-days or man-years with units changing accordingly.

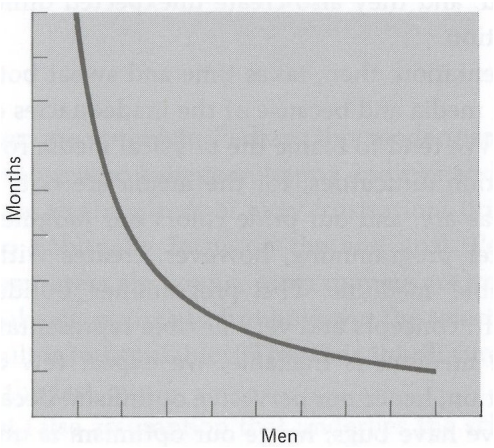

Figure 14 – Behaviour of tasks which can be partitioned in a perfect manner (Brooks 1995, p. 16)

If tasks have sequential constraints, then assigning additional resources does not necessarily have positive effects on the schedule. Brooks (1995) illustrates this issue using the examples of debugging and system test. Since the first edition of his book was released in 1975, debugging technologies have been evolving and allow for a certain degree of parallelisation. However, the author has experienced prevailing sequential constraints for system tests in late stages of the project and modular software development: for example, despite being able to test completed assemblies using mock interfaces, the integration and final test of intercommunication among finished assemblies has to be delayed until development of all components with respective dependencies is finished.

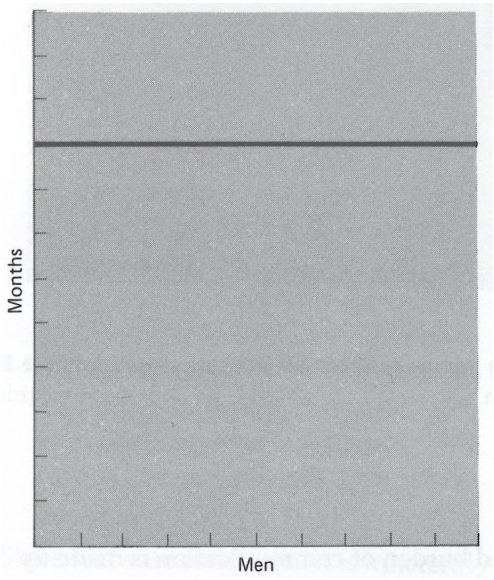

Figure 15 – Behaviour of tasks which cannot be partitioned (Brooks 1995, p. 17)

For tasks which can be partitioned, Brooks (1995) points out the burden of communication among people working on subtasks, which can trigger worse results than what could be expected from an even trade of men for month:

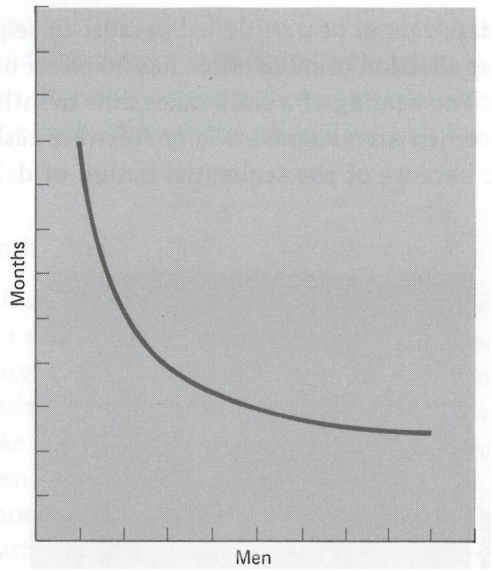

Figure 16 – Behaviour of tasks which can be partitioned but require communication among workers (Brooks 1995, p. 18)

Training is also a factor which can generate more workload when adding human resources. Brooks (1995) cites Vyssotsky (n.d.) to emphasise that training cannot be partitioned and added effort varies with the number of workers in a linear fashion.

Highly dependent tasks can result in a vast amount of intercommunication. “If each part of the task must be separately coordinated with each other part, the effort increases *n(n-1)/2*. (...) If (...) there need to be conferences among three, four, etc., workers to resolve things jointly, matters get worse (...). The added effort of communicating may fully counteract the division of the original task (...)” (Brooks 1995, p. 18).

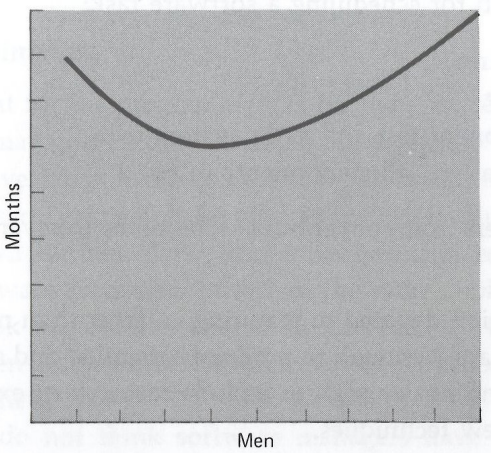

Figure 17 – Behaviour of tasks with complex interrelationships (Brooks 1995, p. 19)

Thus, in some cases “Adding more men lengthens, not shortens, the schedule” (Brooks 1995, p. 19).

3.5.2 The Problems with Common SE Project Planning

In order to sell to a broad customer base, many off-the-shelf project planning applications such as MS Project are designed in a generic fashion. These products can be used to plan for projects in virtually any knowledge area.

When considering the ways in which SE undertakings differ from projects in other fields, it becomes apparent why many common project planning applications have insufficient functionality to plan for SE projects. There is a need for these tools to provide additional specialised functions aimed at the peculiarities of software projects.

Apart from project attributes described in the previous subsection, SE project planning applications need to consider further information concerning e.g. human resources. This can include experience and skills of individuals as well as contingency planning for shortfall during potential times of sickness. While the former factor may enable the project manager to include periods of training in the plan, the latter factor can help to improve the precision for simulation of possible project scenarios and outcomes.

Apart from the option of playing around with certain variables, most generic planning tools do not provide the user with an option to run sophisticated simulations on possible project scenarios. This functionality would be very useful and could be realised e.g. in a sandbox environment: simulations would have no impact on the "real" project plan.

With the provision of virtually no guidance, the usage of common planning applications for software engineering requires a tremendous amount of experience from the project manager if the project is to meet schedule and budget constraints. However, even if the person in charge has many years of experience in the field, two significant problems still remain:

1. The larger the project, the less precise and reliable estimations become if there are no sophisticated planning tools to support the expert in his or her decision making process.
2. Without supporting project plans, certain decisions may be hard to communicate to other corporate departments such as business management and finance.

3.6 General Project Planning and Management Concepts

3.6.1 PRINCE2 vs PMBOK

PRINCE2 (Projects IN Controlled Environments) "(...) is described as a structured method for effective project management for all types of project *[sic],* not just for information systems (...)" (Wideman 2002, p. 1). Furthermore, it constitutes "(...) a widely recognized de facto standard used extensively by the UK government and in the private sector. (...)" (PRINCE2 (n.d.) cited in Wideman 2002, p. 1).
Wideman (2002) compares PRINCE2 with the Guide to the Project Management Body

of Knowledge (PMBOK), which is publicised by the Project Management Institute and can be generically applied to a wide range of projects. He concludes that both PM concepts serve different purposes, which inhibits direct comparison. However, Wideman (2002) reasons that the PMBOK guide covers more ground than PRINCE2 and is more suitable for teaching the subject, rather than using it as guidance for projects. "Nevertheless, within its self-prescribed limitations, PRINCE2 provides a robust easy-to-follow methodology for running most projects, that is, where the objectives are clear and the deliverables are either well described, or capable of being so" (Wideman 2002, pp 8-9).
The statement above can result in the problematic that, especially in the field of commercial SE projects, objectives (i.e. requirements) are not always clear and can often become subject to clarification due to potential ambiguousness. Wideman (2002, p. 9) identifies a similar issue with regard to projects in general: "Considering that it is in the conception and definition phases that the most critical project decisions are made, it is surprising that more focus is not given to this part of the project life cycle both by the [PMBOK] Guide and PRINCE2".

3.6.2 Six Sigma and COBIT

Six Sigma is a measurement of quality and can help to improve the accuracy of certain SE processes. Tayntor (2007, p. 12) states that despite "(...) Six Sigma has its basis in statistical analysis, (...) both the tools and the techniques can increase the probability of successful system development by ensuring that the 'three rights' are in place"

1. The right people are involved.
2. The right problem is solved.
3. The right method is employed.

"Control Objectives for Information and related Technology (COBIT) is a comprehensive set of resources that contains all the information organisations need to adopt an IT governance and control framework" (IT Governance Institute 2007, p. 9). Also, it can help to "(...) optimise IT-enabled investments and ensure that IT is successful in delivering against business requirements" (IT Governance Institute 2007, p. 9).

3.7 Conclusion

The statements in this chapter made by Brooks were published for the first time in 1975. Since then, these ideas have not consistently found their way into widely acknowledged understanding of modern SE project planning.

COCOMO II provides a sophisticated tool for cost estimation through a high level of detail. In the case of COCOMO II however, sophistication resulted in complex usability. As a result, for some companies the model may be too complicated to be considered as a viable alternative to less precise, but more simple cost estimation models. In some cases, the models that are used may rely almost entirely on the experience of the

expert. When using a simpler model however, the estimations can sometimes be easier to communicate and justify to internal business departments and to the customer.

As cited in 2.4.2, progress in the SE discipline is likely to come from building increasingly simple systems and by reducing inherent complexity. This might apply to project planning methods and tools as well. However one should not confuse simplicity with lack of completeness: generic tools may be far from appropriate for SE projects as shown in this chapter.

In conclusion, project planning and management applications need to consider the very nature of software projects if they are to be used for this purpose. This can be realised through tools which are rather sophisticated and take into account certain factors of SE projects, but at the same time provide the user with intelligent guidance and an interface that is easy to use and to understand. As elicited in the preceding literature review chapter, implicit corporate business factors (see 2.9.3) may need to be addressed during planning stages of SE projects.

Chapter Four

The importance of taking into account the nature of software projects when planning for SE is emphasised in the previous chapter. Also, the problematic of generic project planning applications and indications of which factors may need to be considered is explained in chapter three.

4 Implicit Corporate Business Factors

4.1 Introduction

Subsequent sections investigate implicit corporate business factors relevant to commercial SE projects. This analysis has the purpose of shedding light on the degree of influence that these factors can exert on the successful outcome of commercial software projects. The chapter closes with a conclusion describing how consideration of the identified factors could improve the quality of commercial SE projects. Furthermore, the conclusion indicates a potential solution on how to implement this consideration within companies.
The knowledge which is built up in this chapter forms the foundation for a framework to plan for high quality commercial software engineering projects.

Johnson et al 2008 describe the PESTEL (Political, Economic, Social, Technological, Environmental and Legal factors) framework, which can be used to analyse the macro-environments of organisations. A similar concept is employed in this chapter to analyse implicit corporate business factors of software companies.

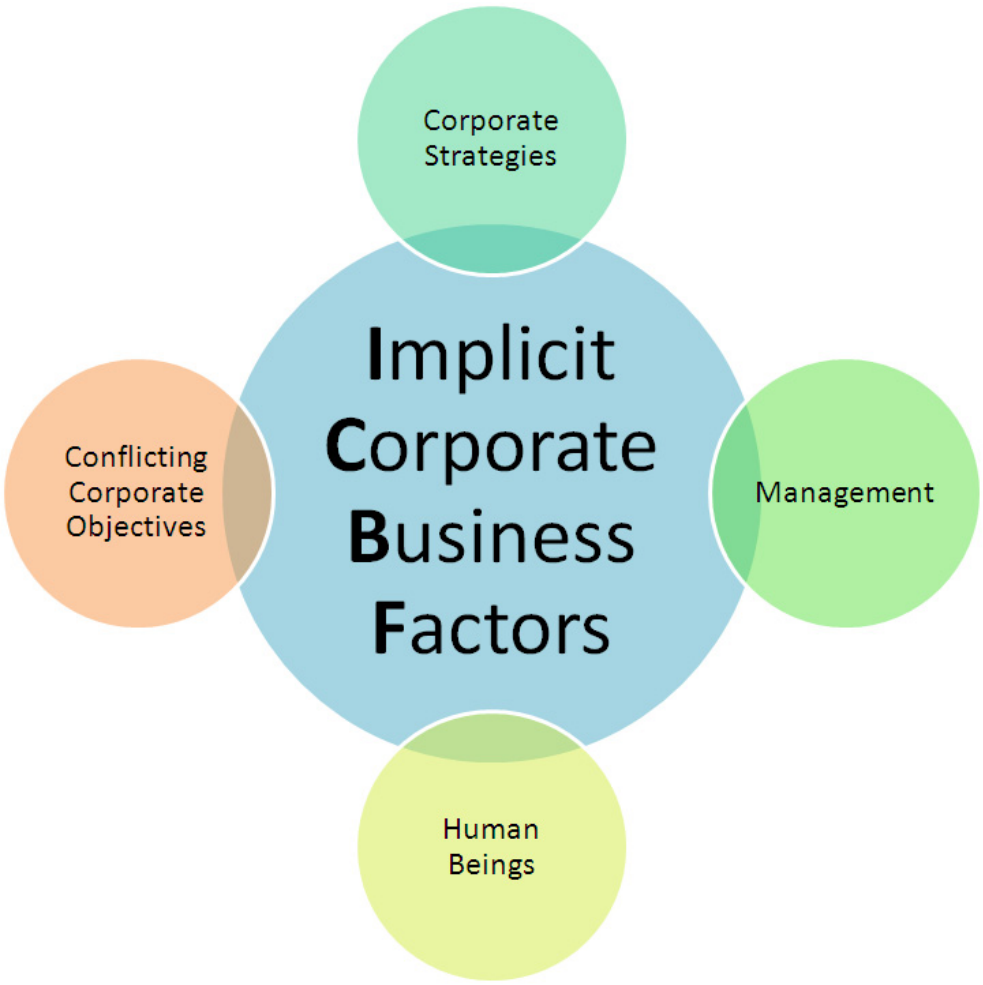

Figure 18 – Implicit Corporate Business Factors (ICBF)

4.2 Factor 1: Conflicting Corporate Objectives

Large companies often consist of many different departments such as marketing, sales, procurement, operation, logistics and distribution. Analysis of corporate objectives in this section however is focused on departments which are potentially relevant to software companies. Furthermore, the analysis predominantly investigates departments which might have the ability to exert vital influence on SE projects. The decision on which departments to analyse is not only inspired by findings from the literature review (particularly sections 2.6, 2.8.2 and 2.8.3), but also based upon the experience of the author in commercial SE projects.

4.2.1 Objectives of Corporate Finance and Senior Management

In their book "Corporate Finance – Principles & Practice", Watson and Head (2007, p.8) describe corporate objectives as follows:

> *(...) employees, customers, creditors and the local community, will have different views on what the company should aim for. It is important to stress that while companies must consider the views of stake-*

> *holders other than shareholders, and while companies may adopt one or several substitute objectives over shorter periods, from a corporate finance perspective* ***such objectives should be pursued only in support of the overriding long-term objective of maximising shareholder wealth.***

Despite Watson and Head (2007) mention non-financial corporate objectives (survival and social responsibility), their discussion of these alternative goals is aimed at maximisation of shareholder wealth and corporate sales figures. Damodaran (2001) raises the question which circumstances would trigger the need for a firm to have share price maximisation as its only objective. He introduces a list of assumptions and concludes that if these proposed assumptions would hold true for a company, then maximisation of shareholder value would be appropriate as its only objective. An example for such an assumption is that maximising shareholder wealth becomes the primary objective for the managers of the firm as well (Damodaran 2001). With regard to the objectives of managers, Byars (1997, p. 10) provides a more neutral description, stating that "A professional manager performs the basic management functions for the ongoing organization". As management is a significant factor on its own, a more thorough investigation of this topic is provided in section 4.4.

Hence, from a corporate finance point of view and from the perspective of senior management positions, the main objective of a firm might be the creation- and increase of value for its owners.

4.2.2 Objectives of Software Engineering

On the contrary to the statements shown above, Glass (1998, p. 11) describes his view as follows: "Software personnel are motivated by things such as challenging projects and/or a chance to use new technology rather than the more traditional ones of power or money" (!)

Some of the aims of other engineering disciplines might as well be applied to the SE discipline: according to Grimson and Kugler (2000, p. 542), "(...) engineering is concerned with creating cost-effective solutions to practical problems by applying scientific knowledge to building 'things' – or systems – in the service of mankind".
The term *service of mankind* however can lead to conflicting interests. Many corporations are run- and owned by people who strive to increase the return on investment (ROI) of their venture, as shown in the preceding subsection. "Serving mankind" could have been the primary intention of the respective founder of the company in the first place, but over time and with changing executive staff, this overall corporate goal can get diluted.

In the experience of the author, most software-engineers aim at producing and delivering high quality software products to the customer and may see this as a reasonable main objective for an organisation which produces software.

4.2.3 Objectives of Human Resource Management

For human resource departments, a central goal must be the well-being of the human workforce of an enterprise. From a Human Resource Management (HRM) point of view, Currie (2006, p. 7) states that "The main purposes of all organisations are to survive and develop". In addition, he argues that organisations are necessary for the infrastructure of modern civilised societies to exist, satisfying the needs of people (e.g. food, shelter, luxury goods etc.). Furthermore, he explains that companies help to handle life situations such as education, health or marriage including birth and death. "Since we are aware of our survival needs of the future, we create organisations to ensure that those needs will be met" (Currie 2006, p. 3).
Armstrong (1992) identifies people as the most important assets of organisations. In addition, he is of the opinion that effective people management is the key to organisational success.

The views presented above clearly contradict statements concerning objectives of other departments. Also, they are an indication that some HRM departments may assume they are compelled to impose their own objectives upon the entire corporation.

4.2.4 Objectives of Legal and Marketing Departments

Legal departments of software companies need to take into account a certain range of legislative aspects. Beside other areas, the law which needs to be covered comprises

- Intellectual property (copyright, patents and trademarks)
- Computer-, electronic contracts and torts
- Criminal law
- Data protection law
- Professional, social and ethical aspects of Information and Communication Technologies (ICT).

(Bainbridge 2008)

Further areas can include the law of negligence in case a software contract needs to address liability for defects and e.g. the status of electronic documents as evidence in criminal trials concerning computer crimes (Bainbridge 2008). With regard to software development, Bainbridge (2008) points out that when drawing up acquisition contracts for computer hardware or software, lawyers as well as computer professionals have to consider the legal implications associated with the technology.

Hence, one of the main objectives of legal departments may be to insure that the corporation complies with the law. Bainbridge (2008) describes the importance of organisations to develop policies concerning the use of computer technology: "(...) systems of auditing should be drawn up to check for unauthorised software, to check for computer viruses and fraud, and to verify that the use of personal data is lawful and in accor-

dance with data protection law" (Bainbridge 2008, p. 4). As a result, other departments might find themselves having to adhere to some or all of the suggested legal policies in a more or less strict manner. This in turn could shift the perception of corporate objectives away from goals related to the software profession towards policy compliance.

Marketing departments may demand the development of innovative products to be able to run advertisements, campaigns or promotions. This could be aimed at a specific product or at the organisation as a whole (e.g. for branding purposes).

4.2.5 Conflicting Objectives: Potential Impact on SE Projects

The discussion of the purpose of organisations leads to differing viewpoints, which almost inevitably causes inconsistency in goals among corporate departments. "Many of the disagreements between corporate financial theorists and others (...) can be traced to fundamentally different views about the correct objective for the firm" (Damodaran 2001, p. 8).

4.2.5.1 Corporate finance and senior management

Managerial decisions which are based solely on short-term generation of shareholder value can have negative effects on high quality software projects. The author has experienced a project where insufficient time was granted to developers for training in a new technology for a particular project. Also, modifications of SE project plans were used for the purpose of vastly increasing business value over a short period of time. The result of these business-deal driven measures was that a project of major importance to the software company was brought to the verge of failure. In this case, the problematic situation of differing internal goals becomes obvious.

In some cases, customers may exert unjustified and exaggerated pressure on senior management, which can lead to uninformed decisions and thus to motivational issues with employees. The reason may be that inexperienced managers abide the attitude "the customer is always right" within the organisation even if the customer is not present. If the customer was wrong in the assessment of the project situation, then technical employees may feel neglected by their leading managers. Therefore, it can be difficult for management to provide superior customer service while making unbiased judgements about the real status of the project.

4.2.5.2 Software engineering

As shown in 2.8.2, overly optimistic time and budget planning may be a major reason for delays in software projects. Sometimes it can be the case that senior management pushes for more workload to be done in a reduced amount of time. Technical people could be tempted to provide plans which are too optimistic as they may want to avoid potentially prolonged discussions with superiors in some cases.
However, the opposite could be the case as well: technical employees could be tempted to avoid the situation of being held responsible for time estimations that may turn out to be too short. Therefore the estimation could be too pessimistic and thus costly. One

further factor which can increase costs could be that some developers might feel compelled to innovate with no regard to the importance of the product. Much time can be wasted on compulsive perfectionism when applied to products which are relatively unimportant to the strategy of the software company or its customers.

4.2.5.3 Human resource management

HRM has similar potential to exert vital influence on other departments as this is the case with corporate finance, top-level management and software engineers: "The corporate culture and the values, organisational climate and managerial behaviour emanating from that culture will exert a major influence on the achievement of excellence" (Armstrong 1992, p. 18). Uninformed interventions by HRM could harm a project or the entire company. In some extreme situations where a project is on the verge of failure, people might need to "pull it off" with working hours exceeding normal standards. HRM should be flexible and intervene with the project plan only if working conditions tend to get out of control for a prolonged period of time[14] in order to preserve the well-being of the employees.

4.2.5.4 Legal and marketing departments

Partial- or even full adaption of compliance rules imposed by the legal department can have counterproductive effects on SE projects, as these rules could negatively impact the morale of the workforce. Without question, legal rules are important to adhere to, but they entail the danger of introducing too much bureaucracy throughout the organisation. Software engineering is partially a creative process, which means exaggerated compliance rules can put developers at unease (see figure 19). This in turn could constrain the motivation, dedication and inventiveness of experts in the long run – a situation which can be worsened if experts begin to develop an exaggerated fear of breaking the law, even if it might be unjustified in many cases of day-to-day operations.

[14] The author has made the experience that friendship and loyalty among team members can become stronger during extreme project situations, which however should not be used as an excuse as such situations should remain a last resort to save the project from failure.

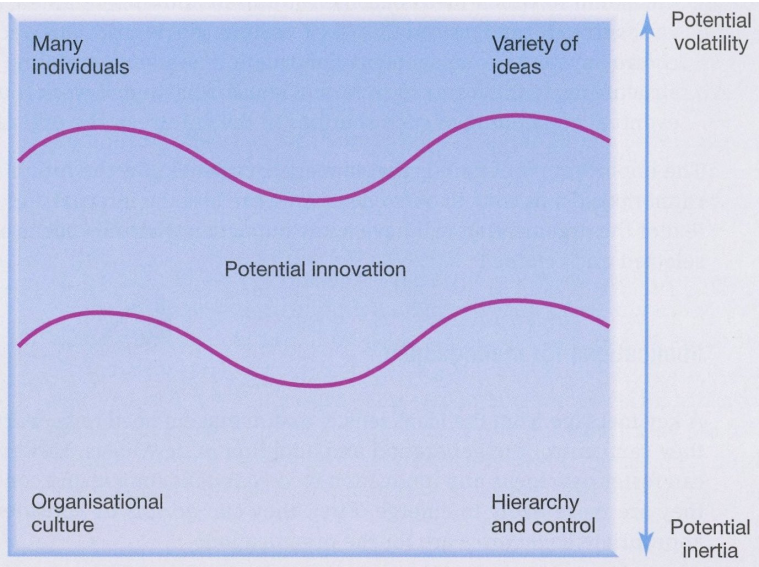

Figure 19 – Adaptive tensions (Johnson et al 2008, p. 40)

Thus, compliance rules may need to accommodate the attributes of the respective enterprise; at least in the approach of legal departments when conveying these policies. Tunkel and York (2000) make the point that technology is developing ahead of the law, which poses a further challenge to legislative compliance rules.

Marketing departments may require innovative products to efficiently build or enhance the brand of the software company. These demands in turn can exert pressure on business and technical departments to sell and implement marketable functionality which may not be needed by the customers.

It needs to be emphasised at this point that each corporate department even *has* to pursue its own and unique goals in order to fulfil its ultimate purpose. However, it appears that sometimes the line gets blurred which separates the goals and aim for the company as a whole from the objectives of individual departments. This in turn can result in a dangerous situation as departments and business units could increasingly act in their own interest – an issue which is presumably more significant for large organisations, because bigger departments could strengthen the feeling of autonomy with employees.

4.2.6 Consolidation of Findings

The deviation of internal goals may sometimes even be apparent to employees and could unfortunately be attributed to a company as being part of normal circumstances, but it poses a serious issue which needs to be addressed: in order for a company to outpace competitors, its individual departments have to work in collaboration. If this is not the case, then decisions which are based on the objectives of only one department may have unforeseen effects on other departments across-the-board. The process of software engineering, which is already highly susceptible to disruption due to its complexity, might be exposed to unforeseen impacts, inevitably leading to higher risk. Hence,

conflicting internal corporate objectives can make the SE process more likely to fail. Reconciliation of differing internal corporate objectives is discussed in subsection 4.6.1 as consideration might form the basis for productive collaboration.

4.3 Factor 2: Corporate Strategies

There is a wide variety of strategies for organisations. In order to provide a relevant scope, this section is focused on major strategic concepts with the potential of exerting strong impact on commercial SE projects.

4.3.1 Industry Forces and Generic Strategies

Porter (1980) describes five competitive forces influencing the competitive state of a company within any industry. For example, two of the forces are rivalry among existing firms in the industry and bargaining power of suppliers. Porter (1980) outlines three generic strategies which can help an organisation to cope with these forces:

1. Overall cost leadership (e.g. aiming to be the cheapest provider of goods)
2. Differentiation (for example, providing products of exceptional high quality)
3. Focus (e.g. on niche markets and particular customer segments)

(Porter 1980)

Strategic decisions which are made to counter competitive forces originating from the industry may have a significant impact on commercial SE projects. The situation can be exemplified as follows:
if a company decides to pursue the strategy of cost leadership in order to outpace a certain competitor, expenses might need to be reduced as much as possible in order to be successful with this strategy. This in turn can result in management demanding from project plans to emphasise rapid implementation of core functionality as well as reduced testing time in order to roll out the system as fast as possible. In this scenario, software engineers could be forced to plan against their motivation of producing high quality.

Therefore, if the strategy of the company is not communicated from the business department to the technical employees in a sound manner, mutual misconceptions concerning the end product may occur with high certainty.

4.3.2 Portfolio Matrices: The Growth/Share matrix

The growth/share matrix (also denoted as the Boston Consulting Group (BCG) matrix) can be helpful in managing the product portfolio of an organisation.

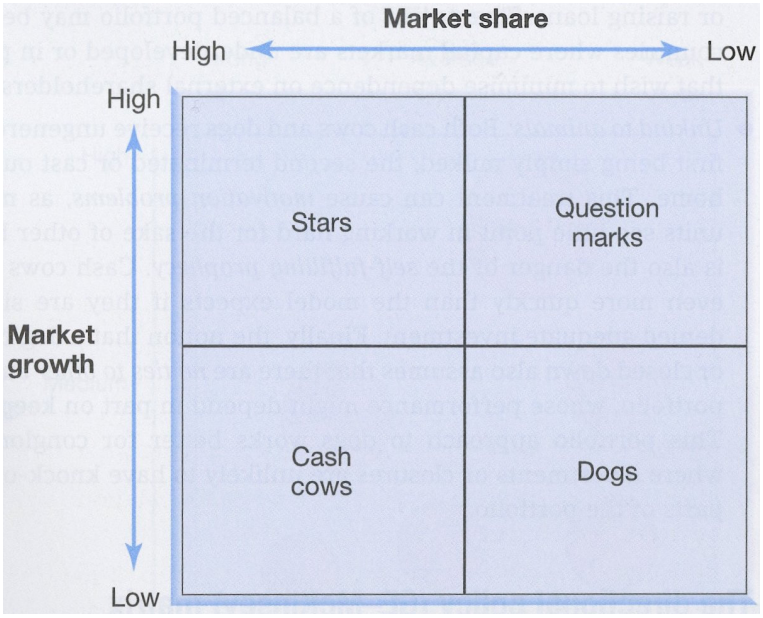

Figure 20 – The growth/share (or BCG) matrix (Johnson et al 2008, p. 279)

Some products in the portfolio with high market share and high market growth (so-called *Stars),* for example, can represent attractive investments. "However, the BCG matrix also warns that high growth demands heavy investment, for instance to expand capacity or develop brands" (Johnson et al 2008, p. 278). Hence, there is a need for balance within the portfolio: because some products could possess low growth rates, the need for investment might be often lessened. *Cash cows* can serve as an example for this type of product: due to its capability to provide relatively constant amounts of cash-flow, this type of product can help to fund promising new undertakings that are e.g. *Question marks.* The resulting investments in turn could aid in turning them into *Stars.* (Johnson et al 2008). *Dogs* on the other hand "(...) may be a cash drain and use up a disproportionate amount of company time and resources. The BCG usually recommends disinvestment or closure" (Johnson et al 2008, p. 279).

Johnson et al (2008) describe a problem with the BCG matrix which is highly intriguing from a commercial SE point of view: The internal treatment of projects which are categorised as cash cows and dogs "(...) can cause motivation problems, as managers in these units see little point in working hard for the sake of other businesses [i.e. stars]. There is also the danger of the self-fulfilling prophecy. Cash cows will become dogs even more quickly than the model expects if they are simply milked and denied adequate investment" (Johnson et al 2008, p. 280).
Strategic planning for software companies might face worse obstacles: software projects which are designated cash cows could not only cause motivational problems with managers of these undertakings, but with software developers as well: the utilisation of individual products in the implementation of corporate strategic moves may be much more difficult to communicate to technical people as e.g. software engineers, analysts and developers than to individual project managers, because the interference in the work of developing software is much more noticeable. Therefore, the impact of prod-

uct portfolio management on SE projects might be much more significant than this is the case with generic strategies described above.

4.3.3 Competitive Advantage through the use of Information Technology

It has to be noted that concepts for generating competitive advantage through the use of computerised information systems sometimes fail to address and emphasise the problematic described above. The reason for this is simple: in the case of strategic use of information systems, the corporate strategy is being aligned to opportunities enabled by IT **and not vice versa.**
However, the concept of utilising IT to enhance corporate strategy shares problems of similar nature with the concept of imposing strategy on software projects: both ideas can create resistance which in turn can be damaging to an organisation: Hammer (1990, p. 112) describes problems when using computers to redesign existing business processes: "No one in an organisation wants reengineering. It is confusing and disruptive and affects everything people have grown accustomed to". These issues can be compared with the potential lack of communication and acceptance when exploiting commercial SE projects to pursue corporate strategies.

4.4 Factor 3: Management

4.4.1 Managerial Skills

Corporate leaders (e.g. senior managers) as well as middle- and supervisory management of organisations can have enormous influence on the successful outcome of SE projects. Talented managers are important when conducting the project in a manner which allows software developers to be creative on the one hand, but keep a certain discipline in order to meet financial goals on the other hand. It can be helpful if corporate leaders of software companies have a certain amount of technical experience or knowledge background. "In practice, management skills are so closely interrelated that it is difficult to determine where one begins and another ends. However, it is generally agreed that supervisory management needs more technical skills than managers at higher level. (...) conceptual skills become increasingly important as a person moves up the managerial hierarchy" (Byars 1997, p. 9).

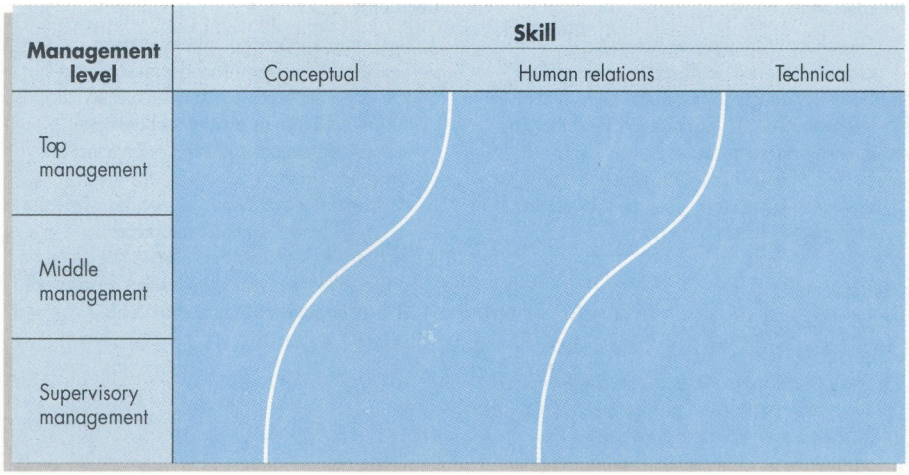

Figure 21– Relative amount of emphasis placed on each function of management (Byars 1997, p. 10)

The previous statement unfortunately does not emphasise that people do not always start their careers at a technical level before they get promoted into management positions. Therefore, it is possible that employees with strong business background, who may have never worked in the field of e.g. programming before, become managers of SE teams. This in turn can create situations where decisions are being made without careful consideration of software engineering principles.

4.4.2 Potential Correlation between Project Types and Degree of Managerial Interference

It is intriguing to note that interference of management with commercial SE projects might be stronger as opposed to scientific or internal R&D undertakings. The reason could be that additional effort to maintain high quality standards seems to be harder to justify for practitioners of commercial SE ventures than this is the case with science projects: whereas the purpose of research to a great extent is often an end in itself, the drive behind commercial SE projects may be profit (views on what the ultimate purpose really is can greatly vary as shown in 4.2). In addition, software for scientific undertakings is often just part of a larger project that may consist of infrastructure such as machines and research facilities as it is the case with CERN[15]. The value of the software becomes more visible as the machinery in many cases cannot operate without it.

Managerial interference could also be less severe during projects for the development of engineering-like systems such as real-time software. Real-time software is often developed simultaneously with hardware components, which could trigger the need to approach the project in an engineering fashion. In this situation, it may be less likely that high-level business management is to interfere with estimations and deadlines during planning stages, because impacts on the quality of the end-product tend to become

[15] For an example of software being part of a large scientific research project, see Stewart et al (2007).

more visible. This partly confirms findings of the literature review, in which invisibility has been identified as one of the major difficulties faced by the SE discipline (see 2.4.2).
Another potential reason why real-time software development might cause business-oriented managers to make rather prudential decisions could be the underlying value of other system parts: real-time projects may be accompanied by the production of actual hardware components such as chips and cables. The monetary value of these materials is – unlike software – touchable and therefore more visible, too. As is the case with research projects stated above, software development then becomes just a part of a larger undertaking.

The important thing to understand is that the value of the hardware components is implicitly projected onto the software. In other words, the hardware would be rendered useless without the software in many cases, just like the machines for scientific purposes described above.

4.5 Factor 4: Human Beings

Dynamics within and among development teams can play an important role during software projects. However, this section is focused on a general analysis of human beings as a factor influencing SE projects.

4.5.1 The "People" – Factor

Parnas (in Fraser et al 2007, p. 1030) states that "The only solution to the never-ending software "crisis" is to try to emulate the science-based, disciplined, document based, development we see in good engineering projects".
Considering results of the research until this point, the author strongly disagrees with this view. Evidence which points towards consideration as being crucial to achieve successful SE projects rather than blindly adapting engineering principles can also be found in recent newspapers:
Elton (2008, p. 2) provides examples of projects where acceptance by people seems to have been a crucial factor to success: He concludes that "These lessons apply to just about every IT project I have worked on – changing the way people work, not just the technology, has to be at the core". In addition, he notes that "No amount of technology will overcome resistance to something people don't want. Attitudes to IT projects are far from being a 'right brain' rational calculation. Success is often about emotion – the 'left brain' instinctive reaction". Thus, if compelling emotional cases are being made, solving technical problems is just a matter of getting work done. "(...) Do it the other way round and solutions are elusive" (Elton 2008, p. 2).

With regard to high-pressure projects and unrealistic schedules imposed on software engineers by higher management, Yourdon (2004, p. 79) makes the following statement: "(...) I suggested that if the project manager can't persuade the customer or senior management to share some of the uncertainty associated with the schedule or

budget of a death march project[16], he should seriously consider resigning from the assignment; the same goes for technical members of the project team". On the one hand, Yourdon (2004) concedes that in the worst case scenario, such a move can imply the problematic of having to look for a new job. On the other hand he argues that it might be a rational and self-protecting move for the employee to recognise that he or she may not be able to bear working conditions which could negatively impact health and productivity in the long run.
Subliminal discontent can become a dangerous threat to SE projects and possibly to the entire company (predominantly for SME).

4.5.2 Invisibility and Mutual Reliance among People

The problematic of invisibility is pointed out in the previous section as well as the literature review: the argument made in 2.4.2 implies that it is difficult to represent software in a geometrical manner. Unfortunately, this may apply to the *planning* of software projects, too. There might be a strong and intriguing link between visibility of the end-product and importance of the people-factor in the successful outcome of SE projects.

The less "visible" and comprehensible plans of products or projects are, the stronger people seem to rely upon one another. This in turn could cause the people-factor to become increasingly important and could be the reason why many IT-projects still get delayed or fail as shown in section 2.4.

4.5.3 Project-Specific Information

The following attributes have been introduced in the previous chapter and need to be included in the factor concerning human beings. Albeit not all of these matters can be denoted as "implicit" factors, they often could be ignored or forgotten in SE project planning in spite of their potential importance. Thus, these attributes are suitable to be included with ICBF:

- Potential shortfall in human resources due to e.g. sickness or training/re-training
- Experience and proficiency in the particular knowledge domain (i.e. whether the individual requires training for the respective knowledge field in which the software may operate).
- Technical experience (e.g. project-specific programming skills).
- General necessity for external competence such as consultants or other 3rd party knowledge input.

[16] *Death march project* refers to a project "(...) for which an unbiased, objective risk assessment (...) determines that the likelihood of failure is ≥ 50 percent" (Yourdon 2004, p. 3).

- Potential degree of required intercommunication among team members and experts.

4.6 Conclusion: Consideration in Planning

The best software process model can be rendered useless if the corporate situation accommodating the SE project is counterproductive.

4.6.1 Consideration

When taking into account the findings of this chapter, one potential solution emerges: consideration.

- Internal consideration: departments would need to increase communication among one another, in order to facilitate their awareness of what objectives, requirements and aims neighbouring departments may have. Internal consideration means reconciliation of different internal corporate objectives.

This includes the consideration for the individual professions within departments of a corporation: as shown in the literature review chapter, the software engineering discipline still seems to be struggling with terminology. The word *engineering* may have led to confusion on how to conduct software development. Consideration of these issues not only for SE, but for all knowledge areas of associated major departments might be crucial. Thus, departments need to consider the peculiarities of the professions of other departments they work with.

- External consideration: the objectives, requirements and needs of individual customers could be taken into account when forming corporate philosophies and strategies. Consideration for the customer can be seen as part of the first factor described above, as different departments may have a different understanding on what is best for the customer.

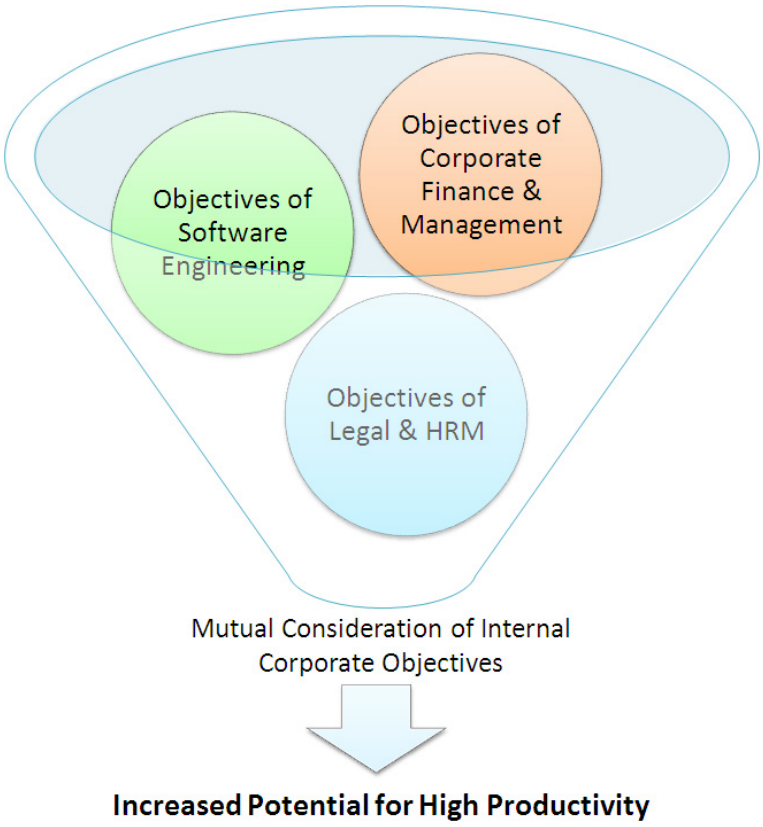

Figure 22 – Potential outcome of improved mutual consideration within a company

A case described by Elton (2008) can serve as an example of the positive influence of consideration: resistance was overcome by delegating responsibility of an IT project to business departments. “The effect was remarkable. (...) Previously insurmountable barriers become challenges to resolve. (...) Of course, IT was a big part of this project, but people played a bigger part” (Elton 2008, p. 2).

However, consideration is not simple to achieve as it creates additional workload and poses the danger of accumulating unnecessary overhead if implemented in the wrong way. For example, meetings to facilitate consideration could result in unproductive gatherings. In the worst case, these meetings may end up with mutual accusations regarding which departments are more inconsiderate than others (which ironically in itself is the opposite of a considerate attitude). How then, could the factors described in this chapter be addressed in order to improve the quality of commercial software engineering projects?

4.6.2 Integration of Factors into Project Planning

One might argue that it cannot be demanded from software engineering to burden the responsibility of handling the factors mentioned in this chapter, as they simply reside outside the domain of software engineers.
However, if these do not get addressed, then nothing will change and it is unlikely that business departments and senior management will take the initiative to improve the situation with difficulties in commercial SE projects. The reason is that, from the view-

point of other departments, these factors might not be the cause of these difficulties as shown in this chapter. Rather insufficient technical project planning models and skills might be identified as a reason for schedule overruns.

The solution may be as follows: instead of directly addressing conflicting corporate objectives, managerial influence and the peculiarities of human interaction, these factors could be integrated into software engineering project plans and thus be addressed indirectly.
If consideration is implemented in an organisation this way, the likelihood of resistance from non-technical departments might be significantly less as opposed to actively attempting a corporate transformation. One could denote this as a "soft revolution".

The author has made the experience in the industry that the planning stage of a software project has the characteristic that consensus can often be reached faster than during any other stage of the project. Emotional reactions during discussions may occur with less severity as usually stakeholders have not invested significant amounts of time, money or effort into the project at this stage.

4.6.3 The Potential Link: Corporate Strategies

Why is consideration (and integration) of corporate strategies important when planning for commercial SE projects?
The answer is that corporate strategies might be able to serve as a link among ICBF **which includes software engineering principles,** because the strategy of the corporation may be something all departments, managers and employees can identify with. After all, strategies need to be understood and promoted by managers anyway in order to be implemented successfully.

Figure 23 – The potential link to reconcile ICBF and the discipline of software engineering

In conclusion, SE project planning in alignment with corporate strategies may significantly increase the quality of commercial SE projects.

With regard to evaluating the intrinsic value of businesses, the U.S.-billionaire and investor Warren Buffett (1983) wrote these lines in his letter to shareholders of Berkshire Hathaway Inc.: “(...) managers and investors alike must understand that accounting numbers are the beginning, not the end, of business valuation”.

Similarly, software engineers and managers alike must understand that SE models and generic project planning tools may be the beginning, not the end of successful commercial software engineering projects.

Chapter Five

The preceding research chapter elicits Implicit Corporate Business Factors. These factors can exert significant influence on commercial software engineering projects and may in some situations be crucial to project success.

5 Planning for High Quality in Commercial SE Projects

5.1 Research Input and Overview

Some existing generic project planning tools comprise an immense amount of functionality. However, these tools may not sufficiently and accurately address the difficulties one encounters when planning for software projects. The key to high quality in commercial SE projects may be planning frameworks which not only consider the peculiarities of the discipline, but also align with strategic objectives and the business culture of the respective software company.
Furthermore, instead of solely using generic project planning tools as e.g. MS Project in conjunction with development tools such as Trac or Bugzilla, intelligent software which is specialised to plan for software engineering projects, might be indispensible to produce more accurate project plans resulting in projects of higher quality.

Intelligent planning software could also provide more clarity for the software project plan. This in turn may allow either corporate department to justify its respective viewpoint in a “visible” manner, underpinned by sound arguments. Also, it can help a department to recognise the potential inappropriateness of its own viewpoints before they are being discussed in time-consuming, “expensive” meetings. As it is the case with the framework, the intelligent planning software may be aligned with (and thus individually configured for) the particular software organisation it is used by.

This chapter utilises the knowledge from the research conducted in the preceding chapters and transforms the findings into practical artefacts. The following sections emphasise the importance of high quality in software projects, as planning for quality forms the basis of the planning framework for commercial SE projects. In addition to the framework, this chapter describes the software prototype for a project planning application. The functionality of the prototype is based on the framework and can support the user in planning for commercial SE projects.

5.2 Research Approach

The framework was created utilising findings of the preceding research chapters including secondary research conducted in the literature review. Therefore, even though it constitutes a theoretical construct, the framework possesses high practical relevance:

input from some literature provides invaluable practical background knowledge about the IT industry. In addition, hands-on experience of the author in several commercial software projects strongly influenced the creation of the commercial software engineering planning framework.

Development of the framework was accompanied by implementation work for the software prototype *IntelliPlan.* Even though conceptual ideas predominantly originated from building the framework, concurrent programming work was useful in that it helped to evaluate not only technical feasibility, but also whether the theoretical concepts would make sense when integrated into project planning software.

5.3 Planning for Quality

5.3.1 Existing Definitions of High Quality Software

Tian (2005, p. 26) states that he adopts "(...) the correctness-centered view of quality, that is, high quality means none or few problems of limited damage to customers. These problems are encountered by software users and caused by internal software defects". Khan et al (2006) provide a rather detailed discussion on what constitutes quality as a concept not only for software but also from a general perspective. They state that "Quality is achieved to the extent that a project and product meets the client's needs and expectations" (Khan et al 2006, p. 3). With regard to a general definition of quality, Khan et al (2006, p. 4) come to the conclusion that "Quality has received different definitions (...) all related to client satisfaction. (...) the definition and meaning of quality is different with respect to different perspectives and there is no uniform, consistent and universally accepted definition (...)".

Taking into consideration the findings above as well as the experience of the author in the software industry, the degree of quality often seems to be measured by determining customer satisfaction due to requirements fulfilment.

Emam (2005, p. 14) uses defects as a measure of software quality: "The more defects, the lower the quality". Measuring quality in this way seems to be suitable especially in case the development is not yet finished, as it can be determined whether the software may meet the requirements of the client in terms of e.g. reliability.

5.3.2 A New Definition for High Quality in Software

It is difficult to estimate in advance which parts of the program might be prone to have errors. Therefore, planning for errors is problematic from a planning perspective. In addition, realistic planning for errors implies that one would have to be pessimistic and negative about the project before it has even commenced. This in turn can lead to overly optimistic (in terms of time, resources and budget) project planning, because only few people may be honest about how many errors could get introduced during development when creating project plans: usually no one wants to depreciate him- or herself at the planning stages of a software project.

But how can one plan for high quality in a project without criteria which may have a negative notion but ultimately enable measurement and validation? The answer could lie within the very definition of high quality in software engineering: it may need to be changed. With many definitions and descriptions of the word quality available as shown in the previous subsection, the author believes that quality of an object can be measured through the amount of consideration which influenced its creation.

High quality in commercial software engineering projects (and therefore in software as a product) may be the degree of unconditional mutual consideration, that is compassion, among human beings who participate in the project. If this criterion is met then other factors might fall in place without the need for explicit problem solving.

This might require effort from the software engineer in that he or she has to become more compassionate, tolerant and empathetic. This additional effort however can pay off vastly because it could result in something positive: enthusiasm. The author also believes there is no better way of delivering quality to the customer than enthusiastic employees, who work hand in hand to achieve this goal; no matter which department they work in.

As indicated in subsection 4.6.2, this *consideration,* which may lead to projects of higher quality, could be integrated into project planning. Imposing "consideration policies" upon the employees of the software company would almost certainly result in resistance and a negative situation, because people would not truly mean what they say: this type of consideration would only be a farce, as real consideration might require people to be compassionate when interacting with their colleagues – including the ones they may not be able to abide.
For companies structured in a way that key software engineers are not members of senior management, the only option for software engineers to induce change of organisational culture favourable to commercial SE projects may be at a technical level: through project planning.

It needs to be emphasised that the initiative might have to come from software engineers (as described in subsection 4.6.2), because from the perspective of other departments, issues with SE projects may stem from problems on a technical level such as software development models, while ignoring that issues could in fact and to a large extent originate from ICBF.
Integration of ICBF into planning might solve the question stated above as it could be way of **planning for high quality** without measuring against factors bearing a negative and potentially discouraging notion such as errors and failure.

Hence, the author is of the opinion that quality *can* be planned in advance through integration of ICBF into the project plan.

5.4 The Commercial Software Engineering Planning Framework

5.4.1 Overview

The framework consists of the following models and concepts:

- Description of solutions provided by the framework
- Stepwise Planning Model
- Generic Strategy Selection Tool
- Strategic Assessment Matrix
- Software Planning Forces Model
- Counterforce Decision Table

The following sections introduce and explain the concepts as well as their usage.

5.4.2 Solutions Provided by the Framework

These four essential difficulties with the software engineering discipline were elicited and discussed in the preceding chapters:

1. Complexity
2. Invisibility
3. Changeability
4. Conformity

This framework addresses the difficulties 1-3 but excludes number 4: conformity might predominantly have to be solved on the level of technical product implementation and planning, rather than higher-level project planning. The four difficulties described above may be sufficient to describe the essence of software. However, in planning for SE projects one essential difficulty which might have not been considered as crucial until date should be added: **Lack of Completeness.** This refers to the problematic, that many generic project planning tools fail to accommodate software-specific attributes as discussed in chapter 3.

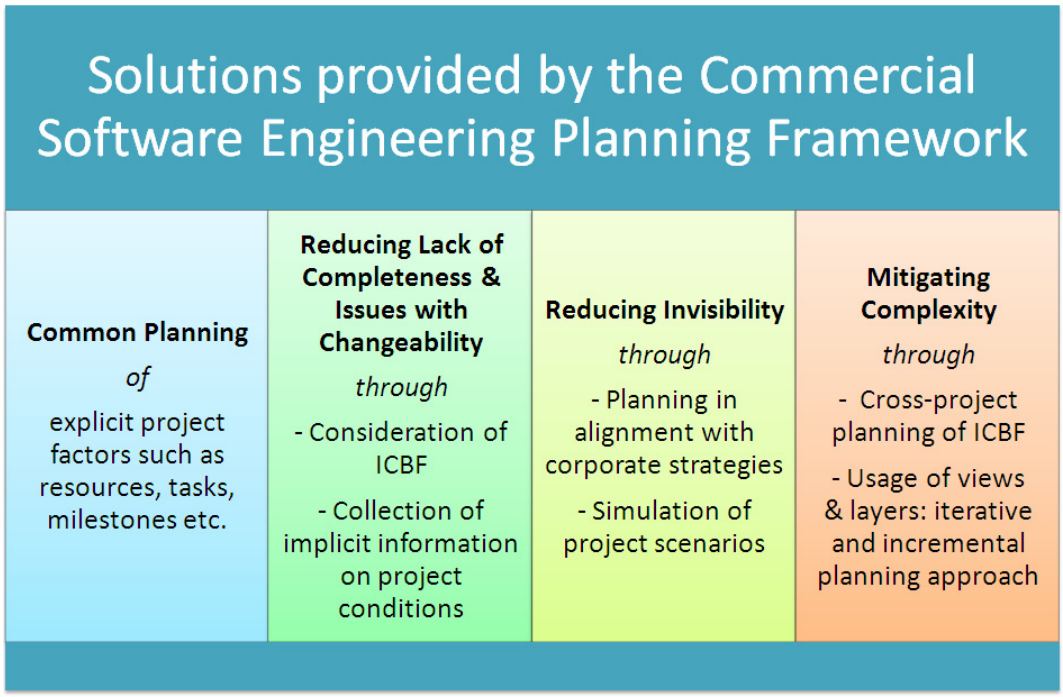

Figure 24 – Solutions for the difficulties with commercial SE planning

Common Planning is essential to e.g. create timelines and allocate resources to tasks. However, in this framework, common planning serves an additional administrative role: common planning is used to quantify (additional) implicit project-specific information and ICBF. **Lack of Completeness and Issues with Changeability** may be addressed efficiently through consideration of ICBF and collection of implicit project information. This might hold true because knowledge about implicit conditions can help to mitigate forces that otherwise would be unknown. **Invisibility** in turn can be reduced by visualising these previously unknown factors and by running simulations on scenarios (supported by software) that include ICBF as well as implicit project conditions. **Complexity** can be mitigated through separating information on ICBF from the individual (common) project plan. Also, building up the project plan in a stepwise fashion improves its soundness and hence makes it more comprehensible (this also applies to views which can be realised in a supporting software planning tool: if required, only "digestible" and understandable fractions of the entire project plan and its influencing forces may be displayed to the user at once).

The explanations above intentionally provide a rather high level overview. Usage and application of the framework is described in more detail in the following subsections.

5.4.3 Approaches for Optimal Usage of the Framework

5.4.3.1 Directional planning approach

The directional approach may be used in case the project planning has not commenced or still resides in its early phases. For the directional approach, the Stepwise Planning Model (5.4.4) can be used. This model resembles a tutorial. Some steps within the

model reference to further models, such as the Strategic Assessment Matrix (5.4.6) and the Generic Strategy Selection Tool (5.4.5).

5.4.3.2 Problem-specific planning approach

The framework can also be used for the purpose of addressing specific or isolated problems in existing, elaborated project plans or for projects which were already commenced. The Influential Forces Model (5.4.7) can help to identify certain problem areas. After detection of potential problem sources, the Counterforce Decision Table (5.4.8) provides structured ways of focusing on a certain problem to solve it with high efficiency.

5.4.4 Stepwise Planning Model

This model provides research information in a concise manner. Entries which have an arrow in front encourage the respective usage of other models described in this chapter.

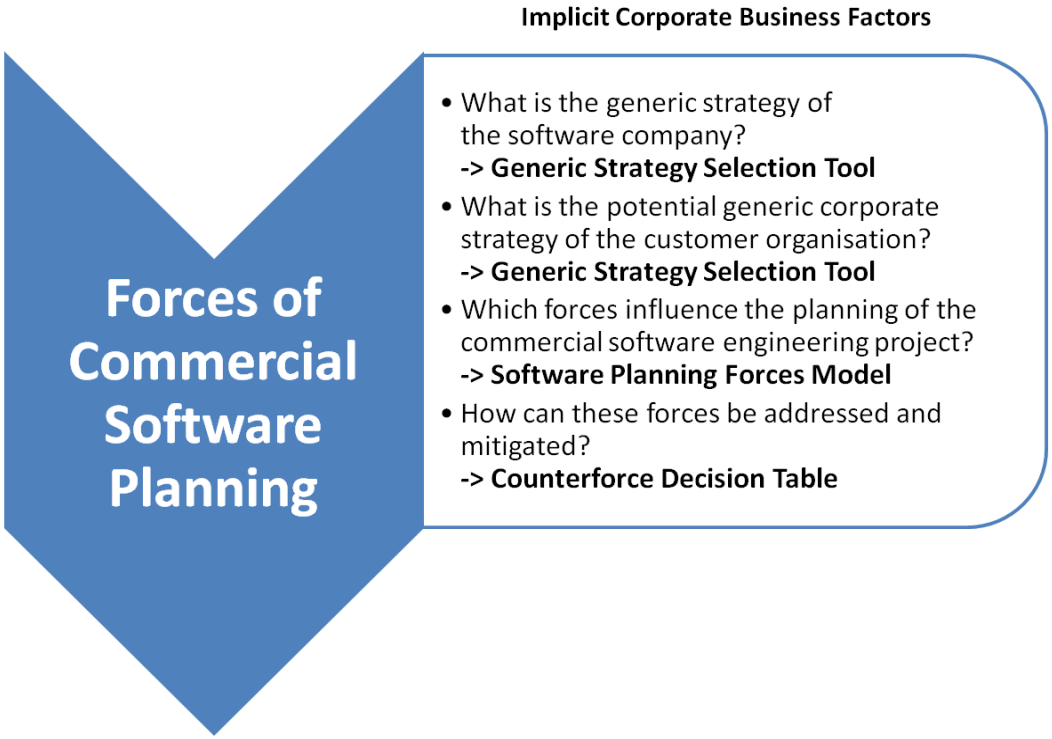

Figure 25 – Elicitation of implicit corporate business factors

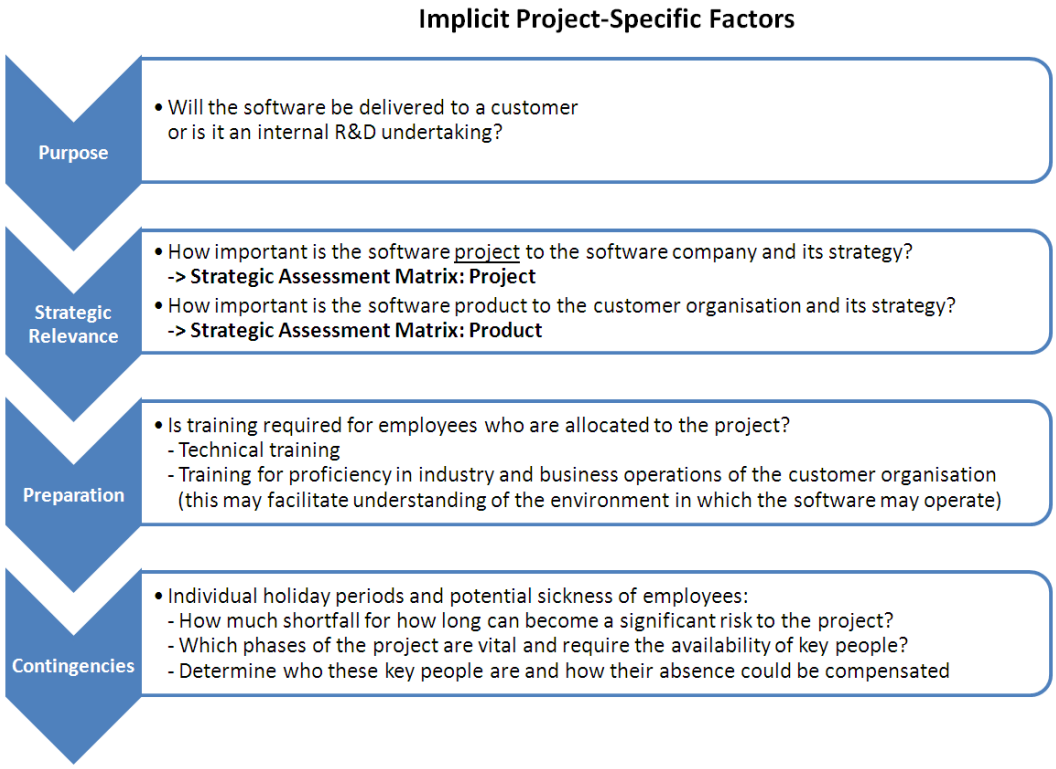

Figure 26 – Elicitation of implicit project-specific factors which coincide with ICBF

"Contingencies" could also comprise potential issues with technical equipment. This however may be categorised under planning for "Resource Requirements" as depicted below in the section for estimation. Due to its technical nature, it should not be part of ICBF.

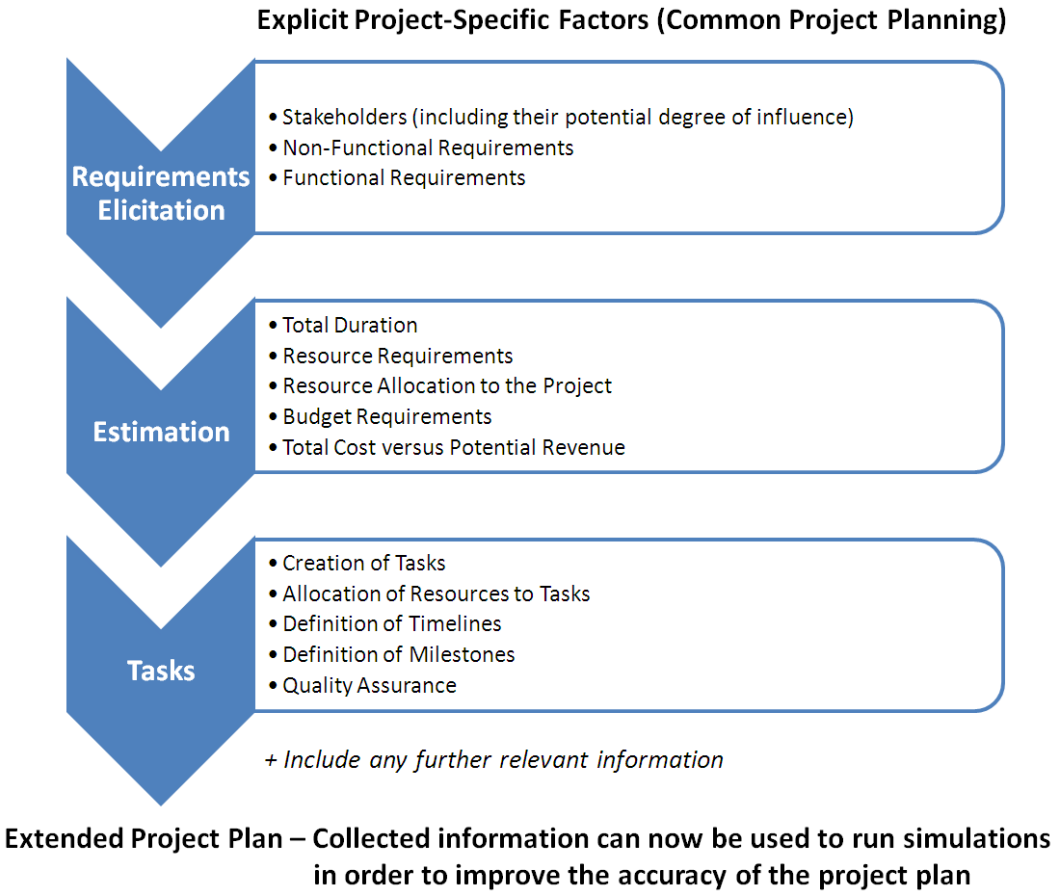

Figure 27 – Elicitation of explicit project-specific factors through common project planning

Concerning the potential influence of stakeholders, the concept of *Influence Maps* (MindTools 2009) may provide a suitable analysis model:

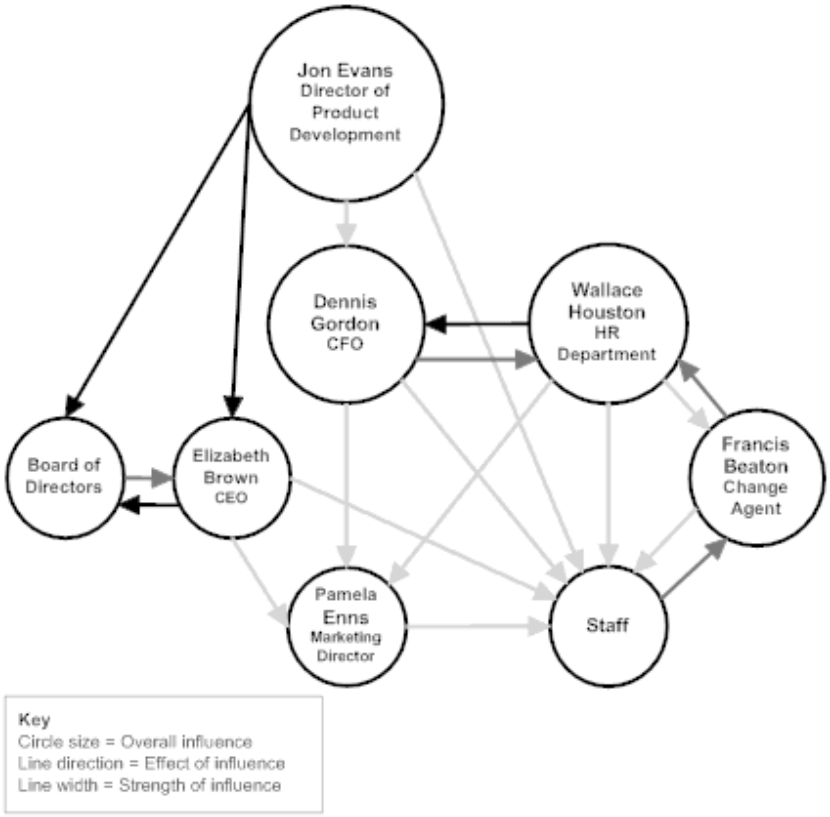

Figure 28 – General example for a stakeholder influence map (MindTools 2009)

Hence, influence maps may be an opportunity to determine the influence exerted by individuals[17] upon commercial software engineering planning.

The concept of running simulations mentioned in figure 27 refers to the possibility of creating different project scenarios with respective approximated outcomes. For this to be sufficiently accurate, simulations might have to be conducted with the support of software project planning tools.

[17] Forces which may emanate from interest groups (i.e. stakeholders) are depicted in subsection 5.4.7.

5.4.5 Generic Strategy Selection Tool

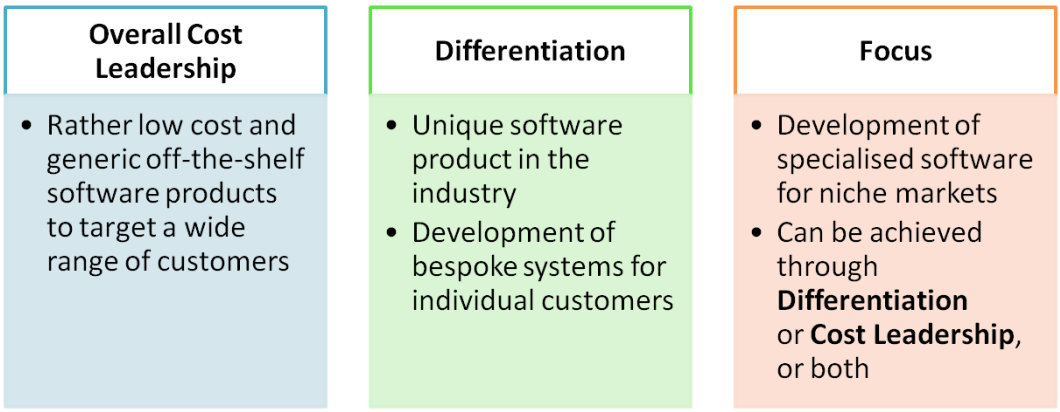

Figure 29 – Generic Strategy Selection Tool for software organisations; the concept of generic corporate strategies was originally developed by Porter (1980)

The Generic Strategy Selection Tool can be helpful to clarify which generic strategy the software company may have chosen to implement. In addition, it can be used to make an educated guess about the strategic interests of the customer organisation. Being aware of fundamental strategic goals forms a vital basis for commercial SE planning, because it provides information on the question which projects or products may fit a certain corporate strategy.

5.4.6 Strategic Assessment Matrix

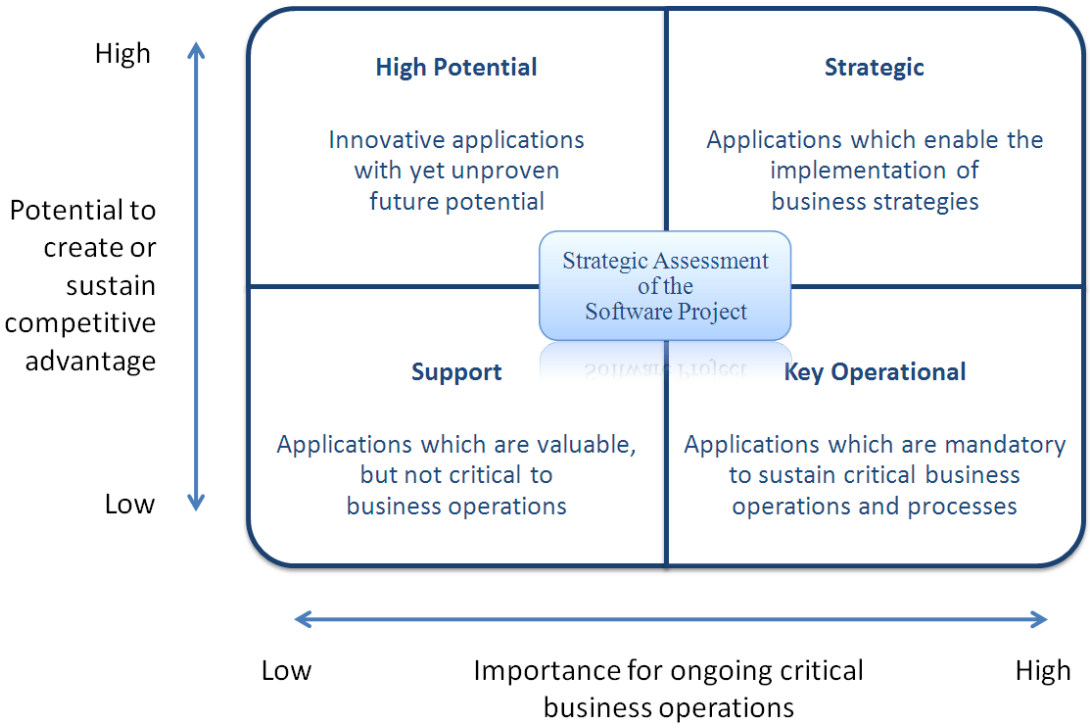

Figure 30 – Strategic Assessment Matrix; adapted from McFarlan (1984 cited in Ward and Peppard 2002)

The Strategic Assessment Matrix can be used to determine the relevance of a software project from two vantage points:

- How important is the software product to the organisation of the customer?
- How important is the software project to the software company?

The "importance" can be measured through identifying in which ways the software product would support the client company in pursuing its aim, objectives and generic strategy (see previous subsection). Also, the software project could be important to the software company in pursuing strategic goals: for example, the software may be reused for further ventures or sold to a larger range of customers in the future and thus could become increasingly relevant from a strategic point of view.
For efficient usage, targets could be placed on the grid to provide a starting point for approximation of the respective percentage. This method may appear as being inaccurate, but it might be sufficient to serve its purpose, which is a sound representation of the potential of the commercial SE project or product.

5.4.7 Software Planning Forces Model

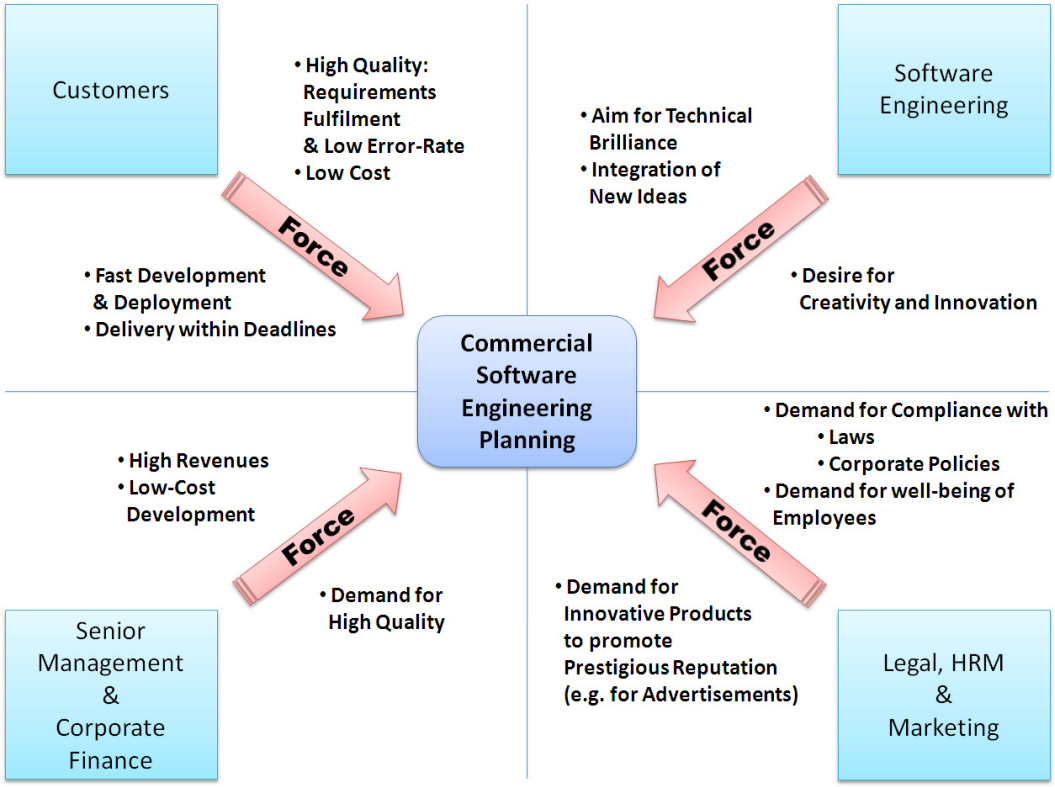

Figure 31 – Software Planning Forces Model[18]

[18] Even though the Software Planning Forces Model is not directly adapted from existing concepts, the idea for this model originates from the infamous concept by Porter (1980) to represent five forces of industry competition.

The Software Planning Forces Model visualises the forces which potentially influence commercial SE planning. Also, it provides some examples concerning individual objectives of the disciplines involved. The model does not require certain actions but can be used as an awareness tool in case a conflict of interests among the individual departments as well as the customer starts to occur. For example, the Software Planning Forces Model can be used as a basis for discussing origins of the respective interest conflict.

A radar chart might be a suitable way of visualising the approximated degree of influence caused by the different forces:

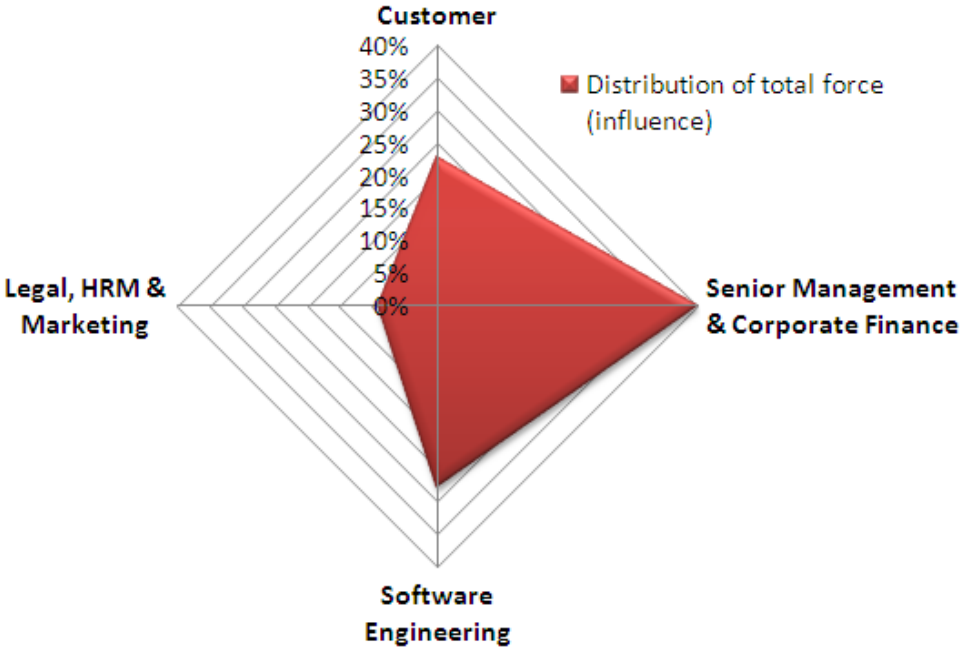

Figure 32 – Example of a radar chart to graphically represent the distribution of potential influence on commercial software engineering planning

When the Software Planning Forces Model is used in conjunction with the Counterforce Decision Table (described in the following subsection), then it can form an important part of planning, not only for individual software projects, but also for improving, optimising and thus strengthening the overall situation within the company.

5.4.8 Counterforce Decision Table

Earl (1989, p. 55) cites the work of Porter and Millar, McFarlan as well as Cash and Konsynski "(...) who have built upon Porter's model to suggest how IT can limit and enhance the five competitive forces for a firm". In a similar manner, the Counterforce Decision table displays the forces acting upon commercial SE planning. Most importantly, it can help to identify appropriate counter-forces to address a specific problem with high efficiency.

The Counterforce Decision Table can be found in Appendix A.

5.5 The Software *IntelliPlan*

The semantics for the software IntelliPlan are based on the Commercial Software Engineering Planning Framework. Practical applicability of the framework may be validated more precisely when using a supporting software application rather than in a theoretical manner through pen and paper only. The software is a prototype and proof of concept and therefore provides implementation of core concepts described in the framework. Despite the conceptual stage of development, IntelliPlan can be used to support common software project planning models and applications.

5.5.1 Design Specification

The software was designed and developed using the following main steps:

1. Design of use cases (see Appendix B)
2. Research into technical feasibility (development of several stand-alone applications to test the programming concepts).
3. Design and development of the GUI.
4. Creation of a conceptual class diagram.
5. Implementation of core functionality (semantics of the Commercial Software Engineering Planning Framework).
6. Implementation of supporting functionality (e.g. data access layer).
7. Extraction of detailed class diagram (see Appendix C).
8. Testing during development was conducted in an informal manner. A description of formal testing procedures can be found in chapter 6.

5.5.2 Details on the Development

5.5.2.1 Choice of technology

IntelliPlan is written in the programming language C#, using MS Visual Studio (VS) 2008 as an IDE. This combination was chosen as it allows for relatively rapid implementation of a prototype. Microsoft .NET framework 3.5 was used rather than developing with an older version, because it provides generic lists which offer better performance than comparable data structures used in earlier versions of the framework.

The usage of Windows Forms in conjunction with Visual Studio 2008 to implement the GUI is useful specifically in prototyping for two reasons:
Firstly, the complexity of event handling of graphical components is partially done by the IDE. Secondly, handling of GUI components (e.g. refreshing and updating contents) requires less effort from the developer than this is the case with e.g. ASP.NET web applications and thus is ideal for quick implementation of prototype software.

Persistence (storage) of data is implemented using XML (eXtensible Markup Language). For the prototype of IntelliPlan, this approach is more suitable than using a da-

tabase for information storage, because the concept does not require the user to store large amounts of data. Hence, there is no need to install additional database software. XML also improves usability in this case as the information is stored in one single project file. Functional tests were conducted using the integrated testing suit of MS Visual Studio (VS) 2008.

5.5.2.2 Development strategy

The development strategy concerns the handling of the VS solution[19]. It comprises intermediate functional tests, documentation, versioning and a suitable backup strategy.

During development, the quality of existing code was improved by several informal functional tests. These tests included debugging of the code in case an error was found: the IDE was used to monitor the contents of certain variables and to assess program behaviour as e.g. branching.
Important classes and methods within the IntelliPlan codebase contain XML documentation which gets automatically extracted and is written into an XML file during compilation. This file can be used to e.g. create standardised documentation files of the code.
The solution received several different version numbers during development in order to support the backup strategy. At the end of every major implementation step, a complete backup of the entire solution was created.

5.5.2.3 Decisions on the program structure

Usage of standardised design patterns[20] facilitates sound practice in software development. The code for IntelliPlan makes use of these patterns:
to control the application, the Model-View-Controller (MVC) and the Observer pattern were used (partially handled by VS). Furthermore, the patterns Singleton and Factory were applied to suitable classes. The MVC pattern allows decoupling of entities, graphical representation and application management layers. The Singleton pattern, for example, insures the existence of only one single instance of management classes such as *ApplicationManager* and *EntityManager.* The factory method pattern is applied to the class *ProjectFactory,* which instantiates (creates objects of) the entities *SEProject* and *Person.* The purpose of the class *EntityManager* is to hold references to major entities: in case components require a factory to create an object of a certain type, then this request gets handled through *EntityManager* to insure consistency in the number of instantiated entities.

The class diagram for IntelliPlan can be found in Appendix C. The diagram provides an overview but excludes design pattern behaviour (e.g. Singleton) and details on rela-

[19] *Solution* refers to the file created via MS Visual Studio holding references to development projects.

[20] See Bibliography: Gamma et al (1995)

tionships among classes, because the design of IntelliPlan is limited to the requirements of a conceptual prototype.

5.5.2.4 Data management and storage

As explained above, IntelliPlan uses the technology XML to store data. This includes the general configuration of the software as well as project-specific information. The classes *SEProject* and *Person* are implemented in such a way allowing their concrete objects to be directly serialised into XML files using standardised serialisation classes provided in .NET 3.5. Vice versa, de-serialisation from XML files into respective objects is implemented in a similar manner. The two methods for serialisation and de-serialisation reside on the data access layer in the class *ApplicationManager.*
The functionality of the data access methods is created in a generic way (analysis of object types via reflection in C#), allowing them to be used for serialisation of not only objects of type *SEProject, Configuration* or *Person,* but virtually any type of object (given it conforms to the requirements for serialisation as e.g. declaring the class public and defining properties to provide the serialisation method with access to values stored in private variables). The approach of writing generic methods (where appropriate) has another positive effect: the code can be reused for other C# applications compatible with .NET 3.5 which may require this functionality.

5.5.2.5 Code optimisation and maintainability

To facilitate maintainability in the source code, the functionality on GUI level is consolidated into further methods (rather than residing in methods which get invoked through event handling). Some examples for such methods are *CloseProject, SaveProject* and *RefreshGUIState.* This approach also avoids code duplications in the program which in turn mitigates the probability of introducing new errors while fixing existing ones.

To further improve maintainability through readability, formatting functionality of the IDE was used, resulting in consistent layout of the code (program semantics are usually not affected if this is done with care). In addition, naming convention for files, classes, variables (Hungarian notation[21]) and methods provides sound information on respective purposes, types and functionality.

5.5.2.6 Graphical user interface

The design of the GUI for IntelliPlan follows the concept of "eight golden rules of interface design" described by Shneiderman and Plaisant (2005), where this was suitable for the prototype. One example is the display of hints for the usage of mnemonic shortcuts. Also, IntelliPlan supports the user with guidance: for example, in case the user forgot to configure the software prior to creating or opening a project, then a message

[21] The *Hungarian notation* (variable name indicating its type) has not been used for all types as e.g. entity classes of IntelliPlan, as this would have made the code less simple to read and understand.

box gets displayed with respective information on how to proceed. After the configuration is created by the user, IntelliPlan continues with the operation requested by the user in the first place.

In order to further increase usability, the order of tab positions (i.e. to which control the cursor focuses on if the user presses the tabulator key) was considered. In case the user enters invalid data, then a red exclamation mark is displayed next to the control which caused the error. This was achieved by using error providers and through the creation of validating text boxes. The validating text box is derived from a normal text box control but extended with functionality to check the entry made by the user against a regular expression.

5.5.2.7 Memory management and resources

Even though C# runs a garbage-collection engine, memory management issues needed to be addressed when displaying Windows forms as dialogs: it is necessary to explicitly dispose the form after the user has closed it because the program has created the window in a new thread and hence is not capable of determining when to remove the reference to the instance of the form.

In addition, IntelliPlan uses resource files to store images, icons and most of the titles, texts and captions for e.g. Windows forms, message boxes and labels. This centralised way of storing descriptive information is beneficial when using IntelliPlan for a different language. Rather than changing all occurrences of descriptive strings in the code, a different resource file can be created which can then be used during compilation. Hence, if a resource file for e.g. the German language is created, then the entire software application can be translated into this language with little effort from the developer.

5.5.2.8 Error handling

IntelliPlan handles exceptions such as erroneous XML files via delegate event handlers (similar to function pointers). After exceptions are being caught and handled on low-level layers such as data access, the exception is thrown further up in the invocation chain. This in turn enables to display error messages which are not only appropriate but also user friendly as technical jargon is kept to a minimum.

5.5.3 Problems and Solutions

Certain challenges were encountered on the way to a functional prototype of IntelliPlan. For example, state transitions to manage GUI behaviour were difficult to implement as well as decoupling of functionality to create layers. One of the most difficult challenges during this project however, was the creation of a generic dictionary (storing data in key/value representation similar to a hash-map) which can be serialised into XML. The problem was solved by collecting information on the Internet about how such a dictionary could be implemented most effectively.

5.5.4 Presentation of the Software

In the context of the research, the purpose of the software prototype IntelliPlan is to prove technical feasibility of the framework. An additional benefit from implementing a practical representation of the framework is increased precision of validation: because the validating experts are not solely confronted with theory but can practically apply the concepts by using the software, they may be more able to come to a conclusion whether they find the concepts to be of intellectual and practical value.

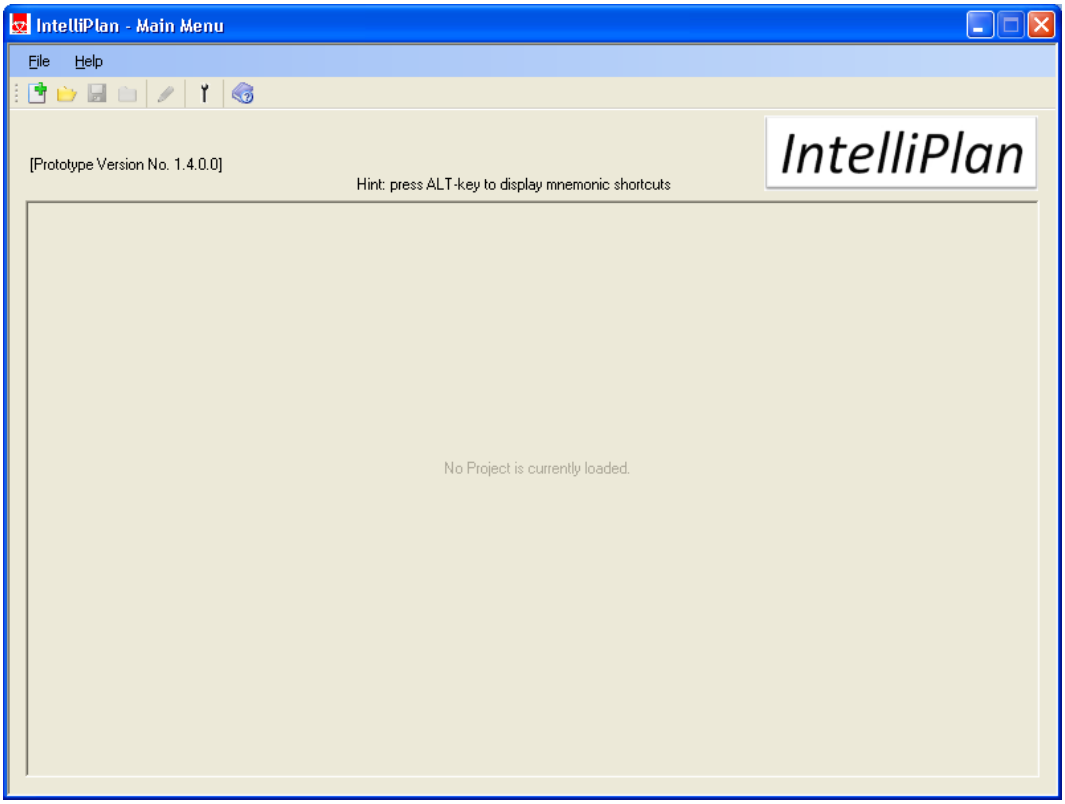

Figure 33 – Main Menu which is displayed when the program is started

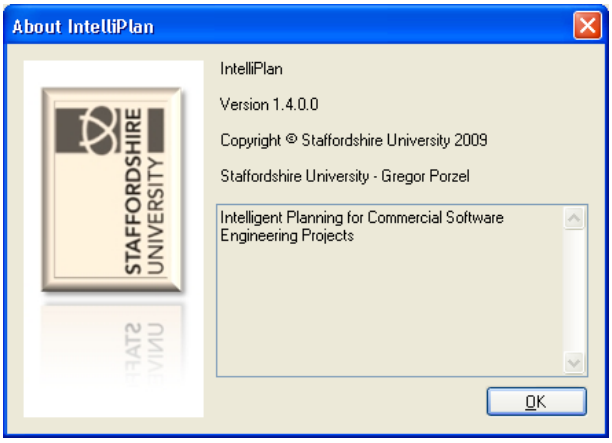

Figure 34 – Program information when the user selects the "About" button or respective entry in the main drop-down menu

If the user attempts to create a new project or opens an existing project file, IntelliPlan may ask to go through the steps of configuring the software. This is usually the case when using the program for the first time: the program checks if a configuration file can be found in the directory of the binary executable (installation directory) of IntelliPlan. If the program has never been configured since its installation or the user has manually deleted the configuration file, then the following message is displayed:

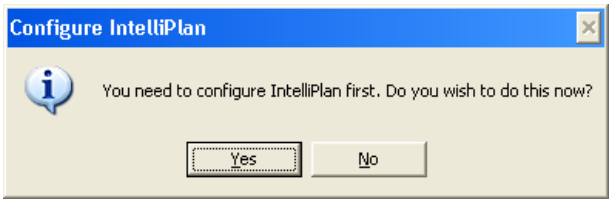

Figure 35 – Message box to ask the user to configure IntelliPlan

The following figure shows the configuration screen which allows the user to input general data concerning the software company. Using the interface, it is possible to set a generic corporate strategy and to add additional notes. Furthermore, the user can define certain technical and business skills with respective proficiency levels for resources and add them to a list. In addition, it is possible not only to remove a resource from the list but also to select and remove certain subentries.

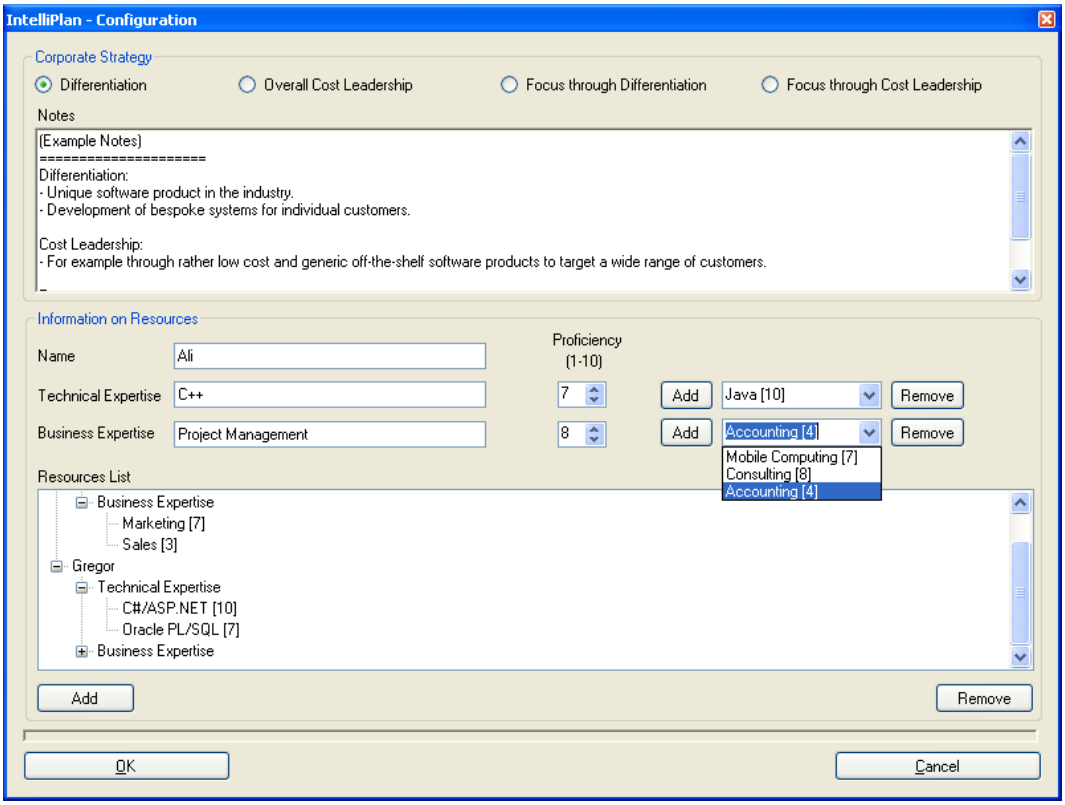

Figure 36 – Configuration interface for IntelliPlan

In case a configuration file already exists and the user chooses to amend the existing configuration by clicking the configure button on the main menu, the application will

ask the user to confirm the replacement of the configuration. This insures that the user does not unintentionally overwrite the configuration.

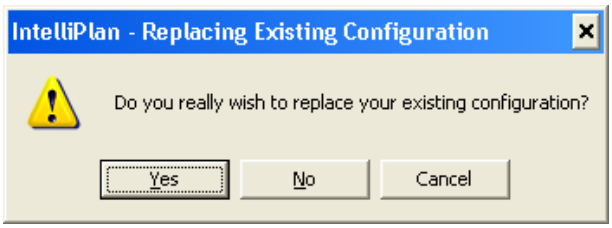

Figure 37 – Message box to avoid accidental amendments of an existing configuration

If the user clicks the OK button on the configuration screen (and clicks Yes in case the message box appears) then the configuration is written to the file "Config.xml" residing in the installation directory.

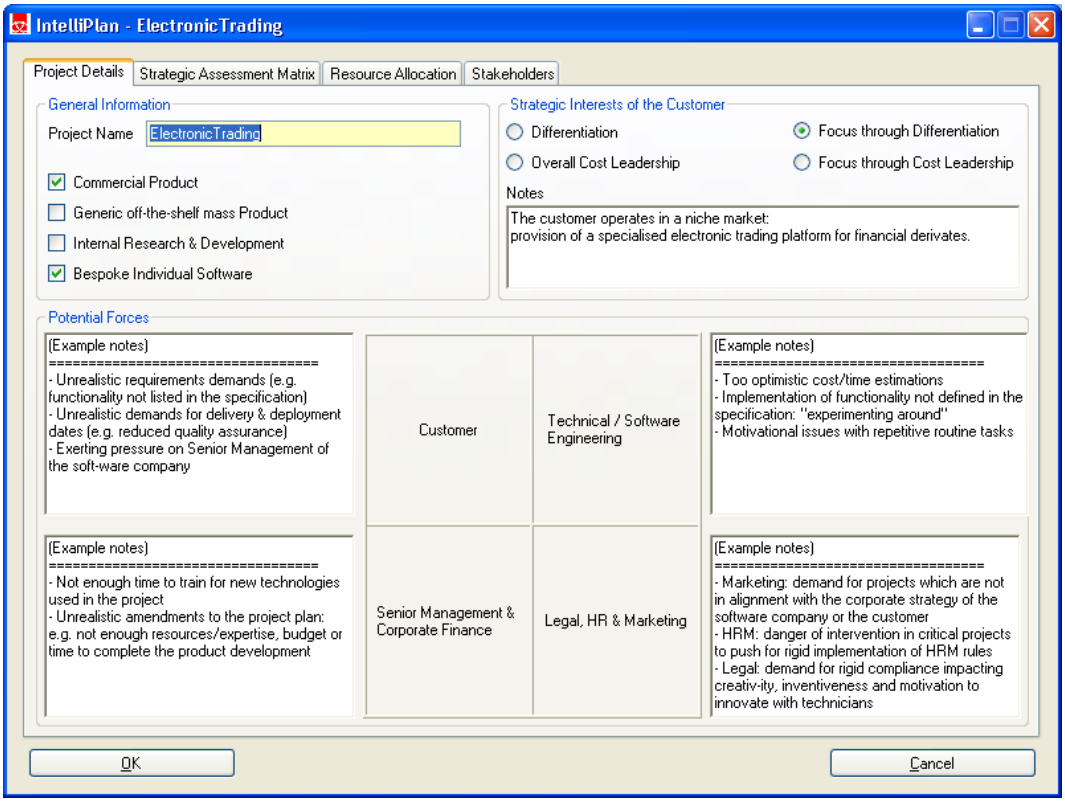

Figure 38 – User interface to create a new project and to edit an existing project

As shown in figure 38, the user can provide the software with project-specific information. If the user edits an existing project, the components (e.g. text box) in all tabs of the form (Project Details, Strategic Assessment Matrix etc.) are updated with respective values to be ready for amendment by the user. In case the user chooses to create a new project, the components display standard values to exemplify their usage.

The following figure shows an example of error handling in case the user enters invalid data:

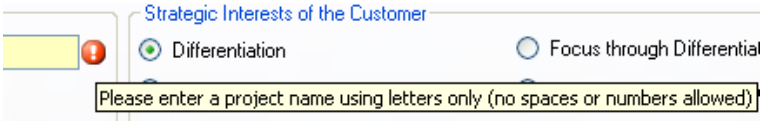

Figure 39 – Error handling of invalid data entered by the user

When creating or editing a project, the user can allocate resources while displaying individual competencies. Upon allocation or removal, the colour of the resource changes in the list of available resources accordingly. This improves the overview as the user can see more easily which resources have already been allocated.

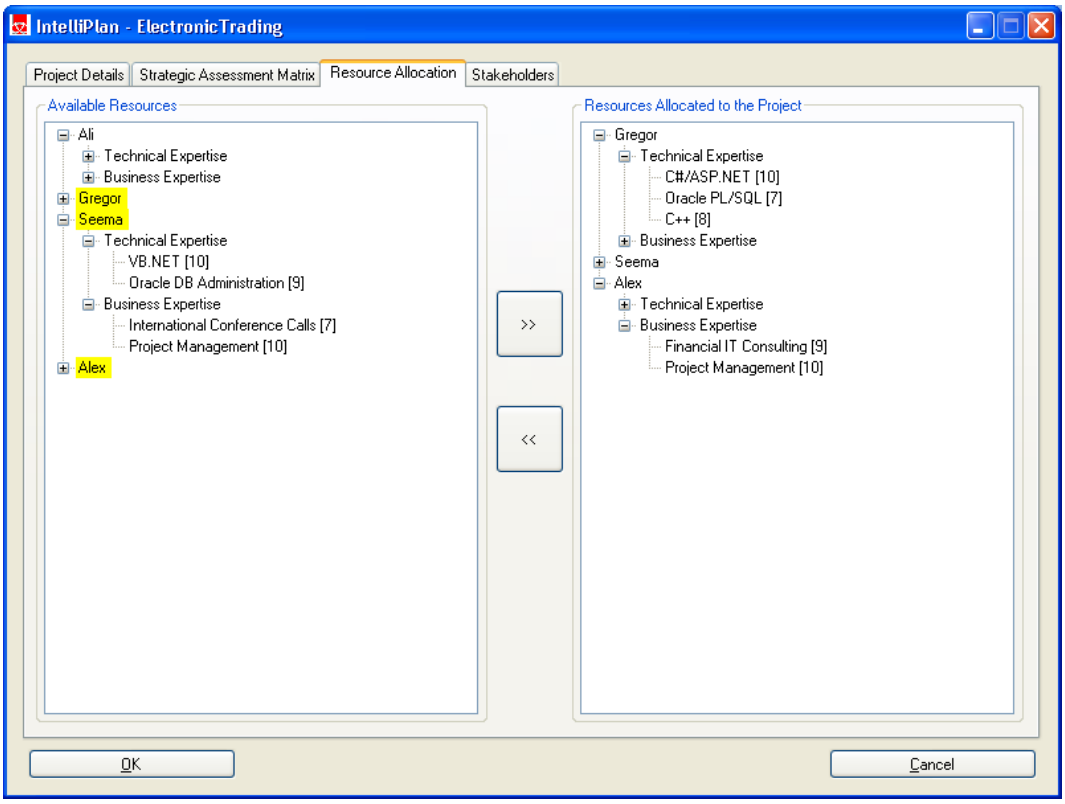

Figure 40 – Resource allocation tab

The tab "Stakeholders" contains a mock-up version of a stakeholder influence map to indicate how this functionality may be represented graphically.

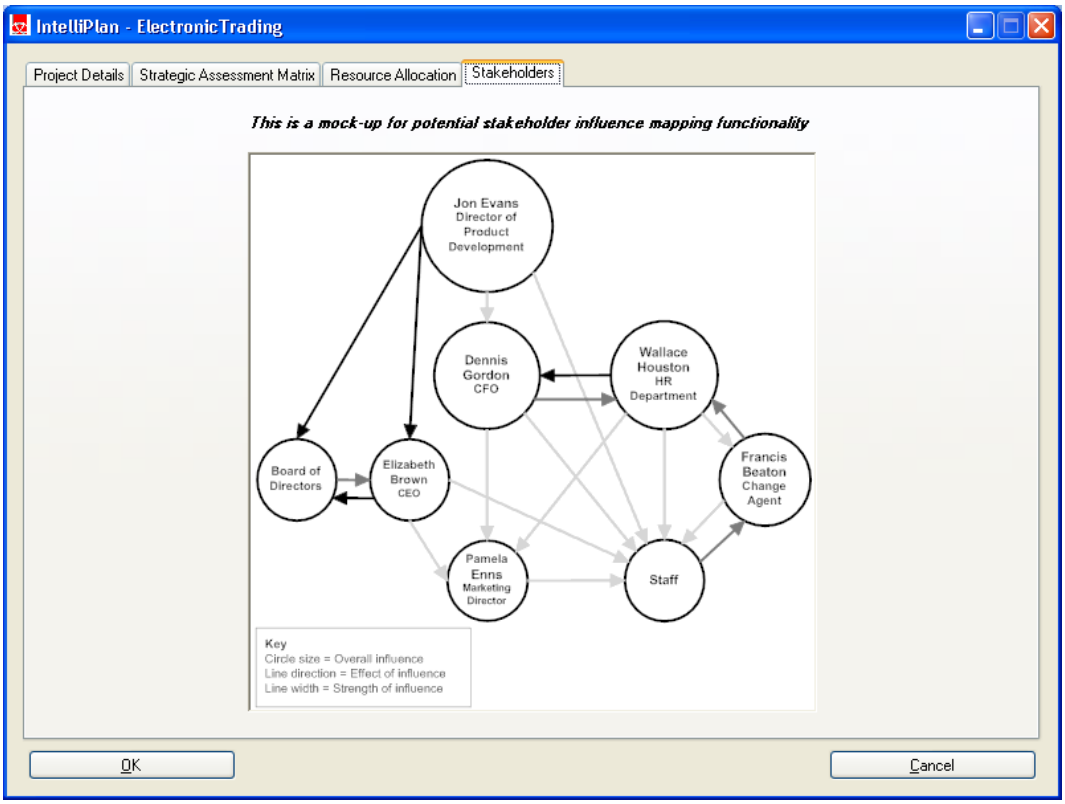

Figure 41 – Mock-up version of a stakeholder influence map; source of inserted figure: MindTools (2009)

If the user clicks OK even though he or she has not yet visited the tab "Strategic Assessment Matrix", then the following message is displayed and the user is redirected to the respective tab:

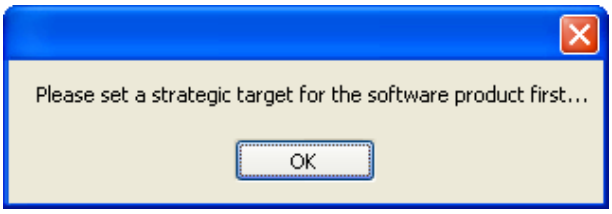

Figure 42 – Message box to remind the user to set a strategic target

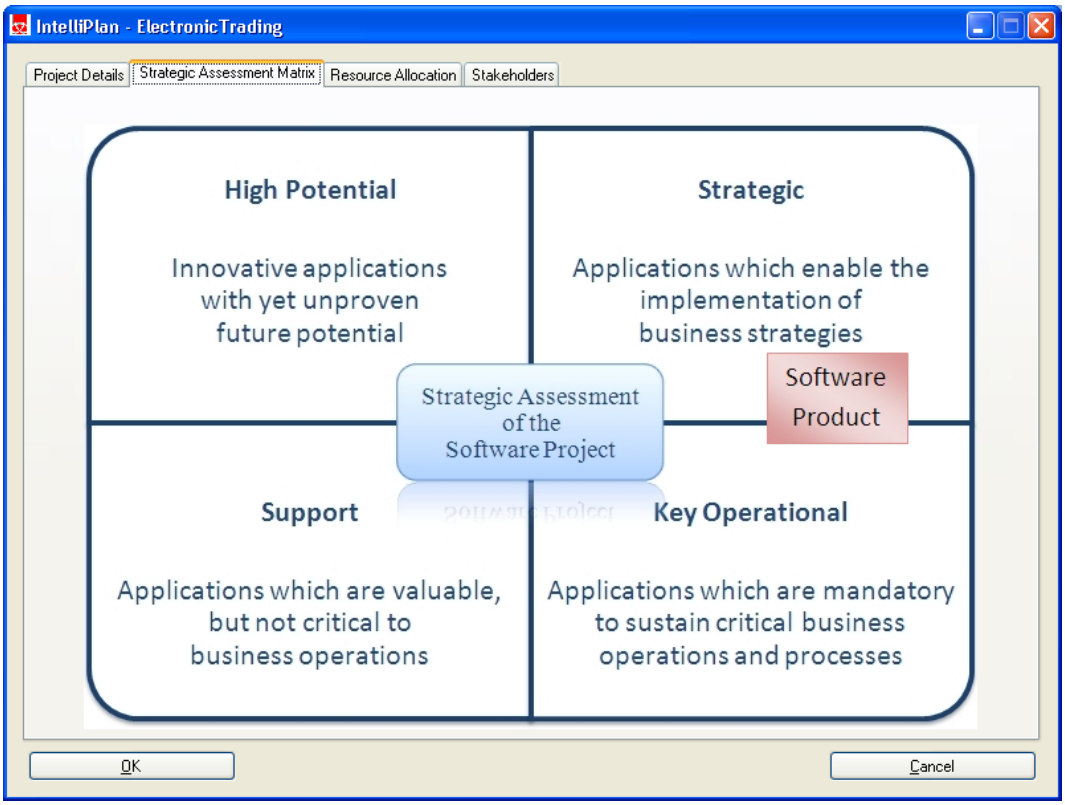

Figure 43 – Strategic Assessment Matrix integrated into IntelliPlan

As soon as the user moves the mouse over the matrix, the cursor changes to a crosshair and a descriptive hint is displayed in the top right corner of the interface. The hint disappears as soon as the user clicks into the matrix to set the strategic position for the software product. In order to amend the location of the software product, the user simply has to click at a different position within the Strategic Assessment Matrix.

Upon clicking the OK button with all required values set correctly, the program closes the create/edit project form and returns to the main menu.

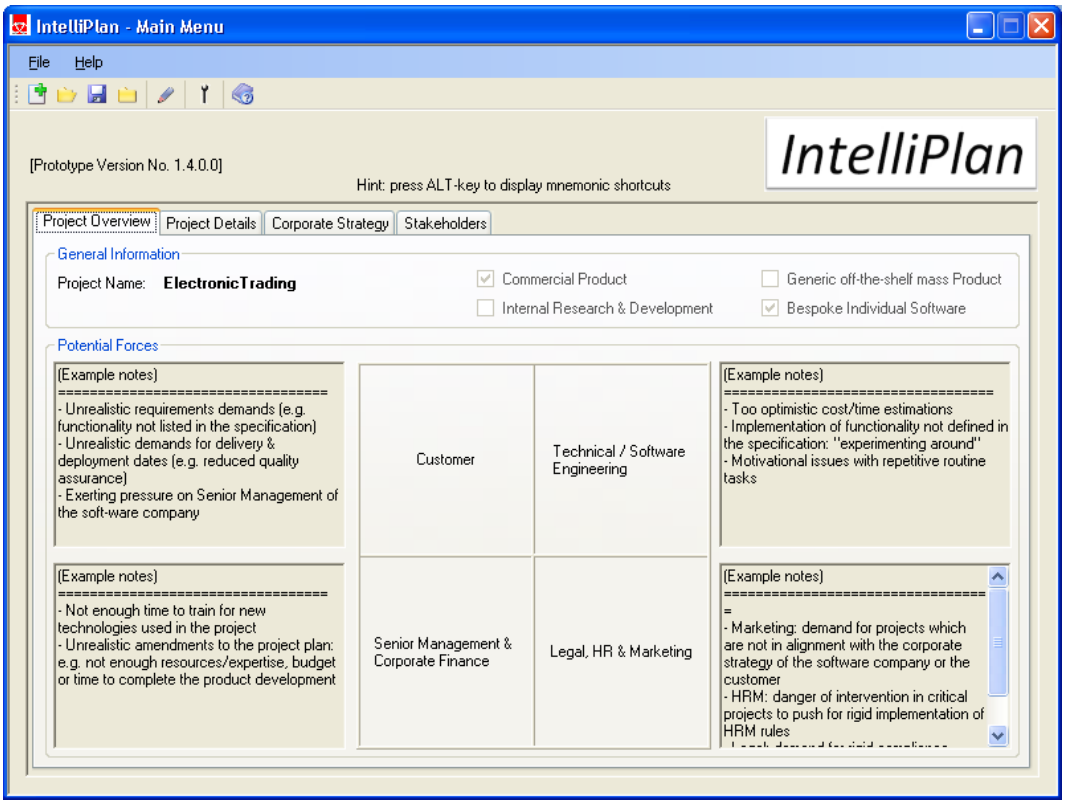

Figure 44 – Main menu displaying a project loaded into the memory

The tabs of the main menu show project relevant information as well as general corporate information defined by the user during configuration. The information displayed on all tabs of this form cannot be edited (to edit the project, the user has to select the Edit Project button in the main menu).

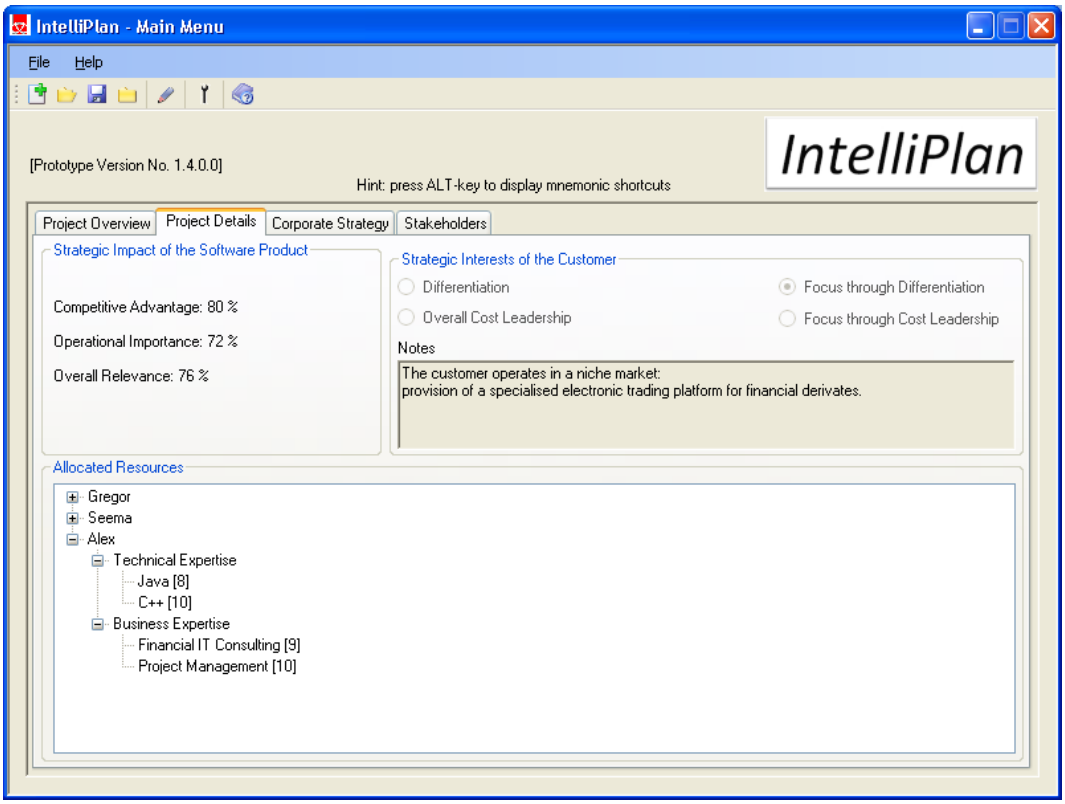

Figure 45 – Main menu: Project Details tab

The tab displaying Project Details contains a percentagewise representation of the values set by the user graphically in the Strategic Assessment Matrix. All other values on this tab are reproduced in the same way the user has provided the data. This also applies to the Corporate Strategy tab which displays the data from the configuration of IntelliPlan as shown below:

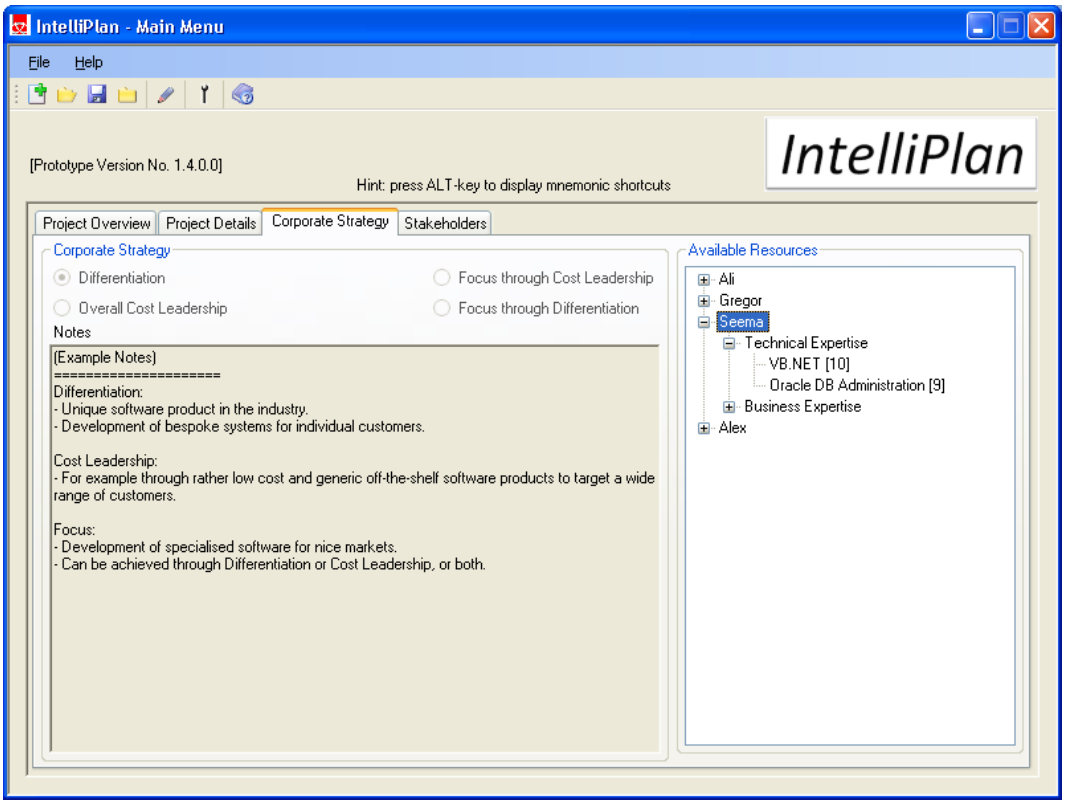

Figure 46 – Main menu: Corporate Strategy tab

(N.B.: the stakeholders tab contains the same mock-up graphic as shown in figure 41). Tooltips provide the user with hints regarding the functionality of the buttons in the toolbar of the main menu:

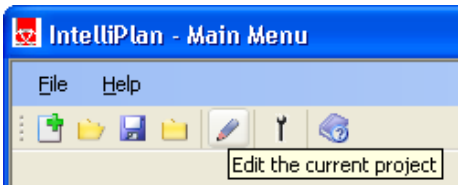

Figure 47 – Example of tooltips for buttons on the tool bar

If unsaved changes are pending and the user attempts to create a new project, closes an existing project or exits the program, IntelliPlan displays a message box asking the user whether he or she wishes to save the changes persistently:

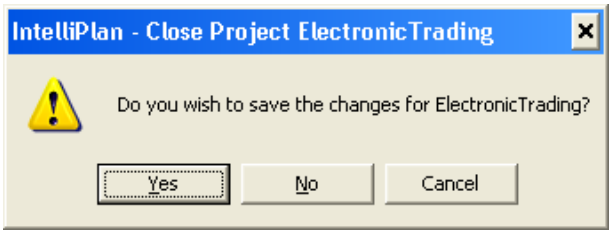

Figure 48 – Message box to ask the user whether the project shall be stored to a file

The user can save a newly created project to an IntelliPlan project file. In addition, it is possible to save an existing project file to a new one (e.g. in order to create a new file for storage of amended information). In case an existing project file in the chosen directory would be replaced because it has the same name as the one chosen by the user, IntelliPlan points out this potential replacement by asking the user how to proceed:

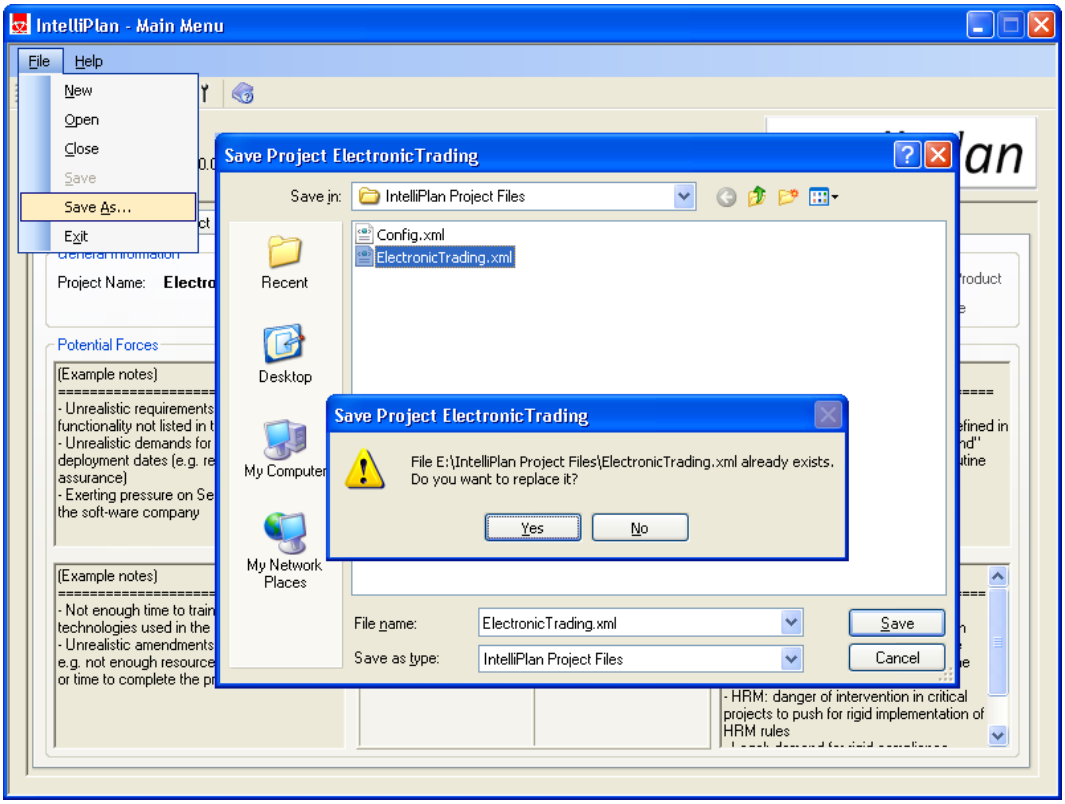

Figure 49 – Message box dialog for saving a file in case the filename already exists in the selected directory

The user can load an existing project file from a data volume using the dialog to open project files as shown in figure 50.

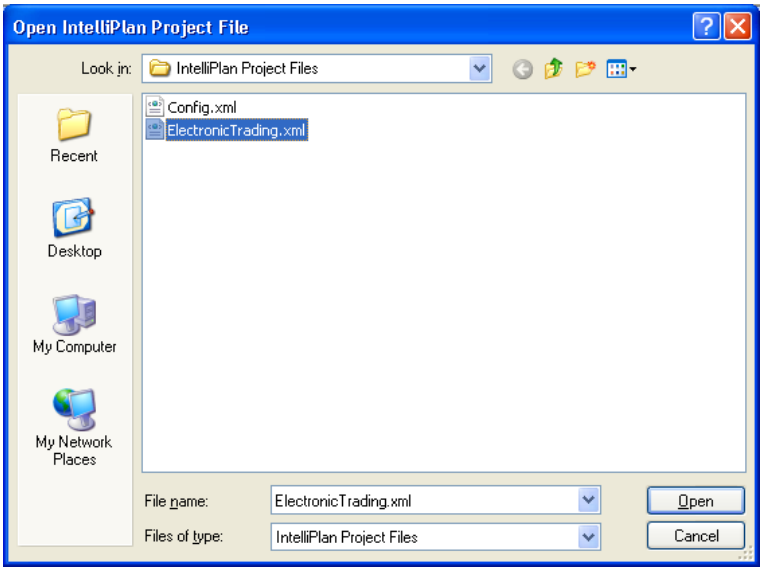

Figure 50 – User dialog to open an existing project file

In case a project file is malformed e.g. because it was modified manually without adhering to the rules for XML formatting for IntelliPlan project files, an exception occurs which is handled by IntelliPlan in order to display an appropriate error message:

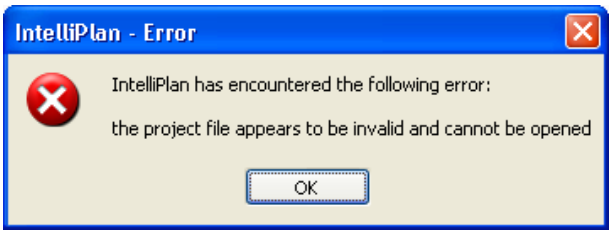

Figure 51 – Error message in case an exception occurred due to a malformed XML file

5.6 Summary and Further Suggestions

The implementation of IntelliPlan has been successful as it provides a demonstration of how to apply some parts of the Commercial Software Engineering Framework in practice. The software has proven important for validation of the theoretical framework as described in the following chapter.

If the software is to be developed further, it should be considered to transform it into a collaborative web-based application. Centralisation of project planning using thin clients also helps to achieve a consistent database. If the software is to be used by external consultants, respective security measures should be in place to prevent from accidently revealing confidential information about e.g. internal strategic decisions.

With regard to the configuration functionality of IntelliPlan, it might be useful to design a stepwise tutorial resembling a digital questionnaire. The questionnaire approach would enable to have the software configured by members of different corporate departments rather than just by one user. This way the software can learn more about aim

and objectives of corporate departments which in turn would enable it to provide better guidance for project planning. IntelliPlan would become more bespoke to the particular company.

Further enhancements may comprise usage of interfaces to AI applications written e.g. in the language Prolog. The AI could support the software engineer in making planning decisions by assessing information regarding the project, the software company and the organisation of the customer. XML data from common generic project planning tools such as MS Project could be imported, which would enable IntelliPlan to be used for analysis of existing project plans.

A significant improvement may be simulation capabilities of different project scenarios as indicated in 3.5.2, 5.4.2 and 5.4.4). The Strategic Option Generator by Wiseman (1985 cited in Callon 1996) could serve as an example of how to structure such a project simulation. ICBF as well as explicit project-specific factors could be summarised in this manner, resulting in a sound visual representation of the project conditions. The AI could then suggest several different outcomes which may be represented using the same model.

Chapter Six

The previous chapter describes the Commercial Software Engineering Planning Framework and the software prototype IntelliPlan. The ideas and concepts for these two artefacts are inspired by research findings of preceding chapters.

6 Testing and Validation

6.1 Overview

The following sections are to insure scientific and practical relevance of the two delivered artefacts through validation. The chapter begins with an explanation of the chosen validation method and a description of functional tests for the software prototype IntelliPlan. Subsequent semantic validation of framework and prototype comprises the opinion of a scholar as well as feedback from industry experts. The chapter closes by drawing a conclusion from input provided from the participants.

6.2 Validation Method

There is a wide range of options to validate artefacts: possible methods comprise e.g. questionnaires, think-aloud protocols, semi-structured interviews and focus groups. However, it has to be elicited which validation method is most suitable for the particular deliverable. In the case of the Commercial SE Planning Framework and the software prototype IntelliPlan, a *Simulated Project Scenario* resembling a case study in combination with an informal protocol for the purpose of recording feedback was chosen as a suitable method to validate both artefacts.
The reason is as follows: research and development conducted in this work is focused on proof of concept rather than details on technical feasibility. Quantitative methods such as questionnaires might fail to capture valuable opinions and ideas potentially arising from out-of-the-box thinking, as participants are experienced experts in the field of SE and thus may provide respective information.

The validation complies with ethical standards and participants were informed about these standards: the persons were asked to read- and fill in an ethical consent form (see Appendix D). In order to convey understanding of the practical context in which both the framework and the prototype could be applied, a simulated project scenario was provided to the participants. The scenario describes a fictitious software project situation and can be found in Appendix E. Notes were taken during discussions with participants concerning the research work, the artefacts and potential application of framework and software prototype to the simulated project scenario.

6.3 Functional Test of *IntelliPlan*

The functionality of the program was tested manually during development. However, some methods can be considered critical and trigger the need for the creation of automated tests. IntelliPlan contains test units for the data access layer:
the class *ApplicationManager* is checked for correct instantiation. Furthermore, two of its generic methods for object serialisation and de-serialisation were tested. These methods represent crucial functionality as they contain the code to handle XML configuration- and project files.
The testing suite which is integrated into MS Visual Studio 2008 is used for the purpose of running the automated tests. This has the advantage that the test project can be integrated into the project solution without having to modify the original source code of IntelliPlan. Furthermore, no references to test-specific .NET components need to be integrated into the IntelliPlan project, as the testing code resides in a separate project configuration. A further advantage is that the test environment is embedded into the IDE which allows for efficient and concise management of the testing procedures.

The expected behaviour of the functionality is defined in the code of the individual test methods. Assertion instructions determine whether a test has passed or failed:

```
/// <summary>
///A test for SerializeToXMLFile
///</summary>
[TestMethod()]
public void SerializeToXMLFileTest()
{
    ApplicationManager_Accessor target = new ApplicationManager_Accessor();
    IntelliPlan.Entities.SEProject serializableObject = new IntelliPlan.Entities.SEProject();
    string fileName = "TestProject.xml";
    bool expected = true;
    bool actual;
    actual = target.SerializeToXMLFile(serializableObject, fileName);
    Assert.AreEqual(expected, actual);
    Assert.IsTrue(System.IO.File.Exists("TestProject.xml"));
}
```

Figure 52 – Code for a method to test XML serialisation for IntelliPlan

The example above investigates whether the serialisation has been successful, not only by checking the return value of the method *SerializeToXMLFile,* but also through investigation whether the file “TestProject.xml” has actually been created.

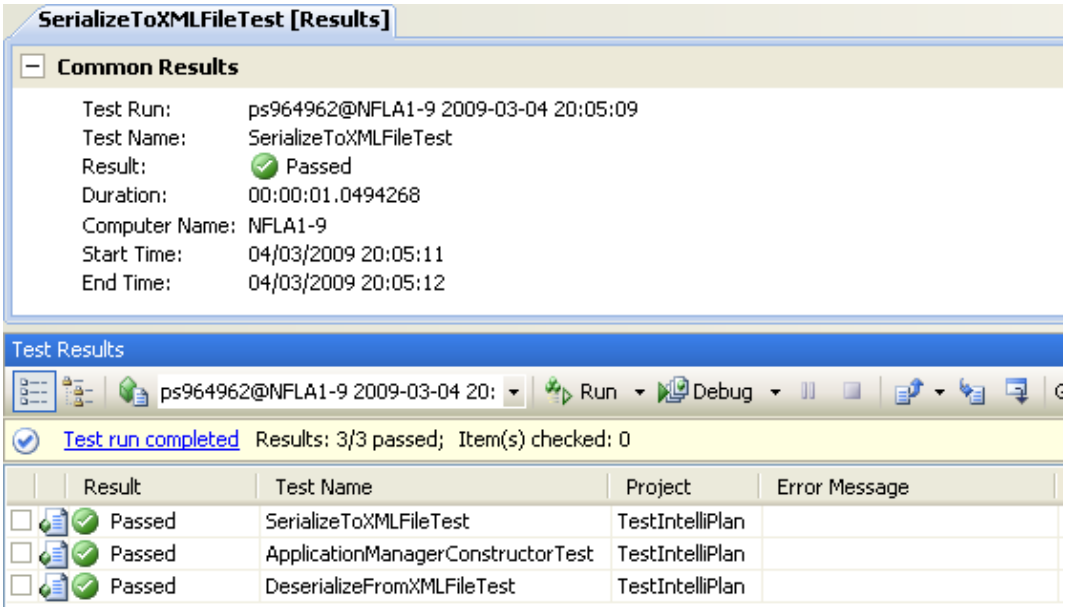

Figure 53 – Test results as displayed in MS Visual Studio 2008

Functional tests of the GUI were conducted manually. Because IntelliPlan is in the development stage of a prototype, the behaviour of the GUI was checked mainly against common sense criteria rather than a detailed specification. The use case diagram (Appendix B) is a helpful mnemonic aid for evaluating the behaviour of the interface regarding its possible states.

6.4 Semantic Validation by an Academic Expert

The participant of the validation appraises the artefacts as follows:
the Commercial SE Planning Framework adds value to project planning. The prototype IntelliPlan is easy to use and shows indications of possessing procedures to guide the user. According to the participant, further development of the software planning tool should focus to provide even more guidance to the user as this may make successful planning less dependent on the experience of the project manager.

Further remarks by the scholar address cross-department usability: IntelliPlan “speaks the language” of business departments and can therefore be very useful for some software engineers to improve their communication skills when conveying ideas to business departments on a less technical level.
The scholar agrees with the research findings regarding generic project planning tools and notes that, for example, in the case of COCOMO II it might not be possible to plan task duration by approximating the number of lines of code which have to be delivered for a certain software development task. The reason is that not every line of code may take the same amount of time to be written as complexity and entropy can heavily vary.

Contingency planning should be linked to the respective strategic or operational relevance of the software product or project. Hence, if a project is of e.g. significant impor-

tance from a strategic perspective, the framework and the software tool need to emphasise the importance of contingency planning to the user.

With regard to the concept of stakeholder influence mapping, the scholar states that connections and communication between stakeholders as well as exertion of influence need to be represented in an explicit fashion. However, the academic expert also points out that only the term "sufficient satisfaction" of stakeholders (rather than just the word "satisfaction") may be suitable as it is seldom the case that all stakeholders can be made "happy" at the same time.

The participant is of the opinion that long-term strategic planning within organisations can be denoted as "programs" such as e.g. the space program. According to the academic expert, strategies have become more focused on aiming for short-term economic goals rather than pursuing long-term objectives. This may have to be considered when developing such a planning tool as IntelliPlan for strategic alignment.

6.5 Semantic Validation by Industry Experts

The industry experts are of the opinion that the concept can fill a void in current project planning for SE projects. They state that implicit factors as e.g. technical skills of developers and potential shortfall due to sickness of persons need to be integrated in the project plan. Therefore, in the opinion of the participants, the framework is ideal in that it expresses implicit facts in an explicit, but simple way. With regard to capturing skills of resources, the experts note that assessment of individual skills can be wrong when it is left to the discernment of the respective project manager.

The forces model was considered as a helpful tool to induce a discussion in the company on basic factors influencing a project.

The industry experts note that the framework and the software IntelliPlan would be useful for planning of bespoke systems rather than off-the-shelf products. The reason is as follows: during bespoke commercial SE projects, the customer tends to control time and material. This is especially the case if a customer company has its own IT department. According to the validating participants, justification of the budget for the project or individual items can be vastly improved by using this framework and its strategic components in particular. Off-the-shelf mass products may be developed internally and expenses for research and implementation might therefore be indeed a subject to discussion, but justification from a strategic vantage point might become less relevant. In other words, the company aims to build a certain product which could yield revenues in future sales "anyway", whereas customers of bespoke systems may tend to be insecure whether the money is being spent wisely.

One expert comments that the framework can be particularly helpful in planning for fixed-cost projects – as described above for the reason that it might help justifying development expenses. He also states that the framework may be less relevant to planning for time-based change requests (financially chargeable amendments of the exist-

ing system). The reason could be that some of these customer requests might be too insignificant to have a strategic meaning.

With regard to pressure of customers on senior management, the experts emphasise that this problem usually does not occur until development is already commenced. Therefore, it can be difficult during planning stages to anticipate the occurrence of such a force.

The validation participants disagree with the negative notion of influence by legal departments stated in the research: legal aspects and policies can contribute to the quality and motivation of the customer and thus have a positive side.

The industry experts were asked if it would be a good idea to extend the planning tool in a way that it incorporates artificial intelligence and possibly data mining functionality in order to guide decision making of the user. The participants note that such an application may be the only way forward to improve existing SE planning tools. However, they suggest that long-term tracking of estimations can become a double-edged sword as this can lead to overly optimistic project planning. Project managers could be afraid that they accumulate a digital history of estimations which turned out to be inaccurate.

Concerning the prototype, the experts are of the opinion that IntelliPlan conveys the framework in a suitable manner. If further development takes place, it should emphasise on more intensive guidance of the user in his or her decision making.

One of the experts suggests that when considering the work from a higher vantage point, the concept of the framework and the supporting software tool does not have to stop with the IT industry. Virtually any industry has to deal with implicit planning factors and corporate strategies.

The participants state that if the framework would be integrated into a fully functional planning software providing guidance and compatibility with file formats of existing generic project tools, IntelliPlan is a product they would be willing to buy.

6.6 Conclusion

All participants consider framework and prototype to be innovative and useful. An interesting finding elicited during the validation is that the importance of software guidance in decision making has been emphasised by the scholar as well as by the industry experts. Therefore, even though participants deem the prototype to be easy to use, potential for improvement may lie in implementing a system which guides the user through all sorts of project planning situations – this could be realised e.g. by creating a wizard as well as electronic planning tutorials.

Chapter Seven

7 Conclusion and Future Outlook

7.1 Key Contributions

The literature review helps the reader to better understand why software engineering may be problematic in some areas compared to other engineering disciplines. The contrasting and comparing of books, conference papers and proceedings, Internet sources, journal and newspaper articles, transactions and research papers has yielded new insights into peculiarities of software engineering. Furthermore, the review undertaken helped to partially describe the discipline from a non-technical perspective. This in turn made it possible to elicit a potential source of issues which in detail might have not been considered to date: the degree of influence external factors can exert on a software project. These factors are identified and denoted in this work as Implicit Corporate Business Factors.

The research following the literature review contributes valuable insight into potential shortcomings of current project planning. Also, description and investigation of ICBF can help the reader to better comprehend organisational forces acting upon a commercial software project. ICBF form the basis for creation of the Software Engineering Planning Framework and the prototype IntelliPlan.

A sound validation of these key contributions through three experts in the field of software engineering helps to insure the academic and practical relevance of the findings.

7.2 Reflection on Achievements and Research Answer

The research on software engineering discovered new aspects of the discipline and allows the reader to understand this knowledge field from a different perspective through technical- as well as non-technical viewpoints. Value is added to the body of knowledge especially by providing opportunities to improve project planning. The production of artefacts turned out to be invaluable to convey the research findings in a concise manner and to allow for efficient validation of the work.

The research question stated in the introduction and literature review is formulated as follows:

Is recognition and consideration of implicit corporate business factors and integration of these factors into planning for commercial SE projects a potential key to successful software engineering projects?

When taking into account the outcomes of the validation of the deliverables, then the research question can be answered with a sound “yes”. Findings of this work show that project planning for commercial software engineering may need to break through the scope of being focused mainly on technical aspects: forces exerted by external factors to SE projects differ from other disciplines such as construction engineering, simply due to the nature of software as shown in this work.
Therefore, it is vital to consider these forces in project planning. The validation has clearly shown that the Commercial Software Engineering Planning Framework as well as the prototype IntelliPlan, which are both based on research findings such as ICBF, may indeed be a key to more successful projects and thus higher quality in commercial software engineering.

7.3 Potential Commercial Value

The industry experts stated during the validation that if the software prototype IntelliPlan would be developed into a more sophisticated software application, they would be willing to buy this product as they found it can add value to their project planning. Recommendations of the author concerning further development of this planning tool can be found in section 7.5.

7.4 General Strengths and Weaknesses

One of the strengths of the research findings lies within its relevance: the analysis of software engineering and external factors may help the reader to better understand the potential origin of problems with software projects. As many issues with commercial SE projects prevail to date, it is important to provide investigation into these problems.

The major strengths of framework and prototype lie in the potential reduction of imprudent intervention in software project planning. In addition, the artefacts can help to justify budget requirements to management and customers and thus can make life easier for software engineers. The validation found that both framework and prototype are easy to understand and to use. This achievement was important to the author as he believes that sound and understandable common-sense planning methods are crucial especially when planning for large and complex software systems.

The framework may be improved in that it could be extended to consider more economic and industry factors in order to become more accurate, but these additional attributes need to be added through layers without introducing too much complexity.

The prototype was helpful in validation of the framework but would need to be more sophisticated in order add value to a real commercial software venture.

7.5 Recommendations for Further Research and Development

It may be worthwhile to conduct research into how other knowledge fields manage to pursue their interests despite prevailing forces in organisational environments. For example, Armstrong (1992) describes problems, approaches and opportunities when integrating business interests and human resource strategies. Similar research may exist for other disciplines and could be taken into account when further developing the Commercial Software Engineering Planning Framework.

As mentioned in section 5.6, AI could be utilised in supporting the software engineer with guidance for decision making during project planning. This opportunity should be considered when conducting further development on IntelliPlan. In addition, the planning software might serve its purpose best if it becomes a web-based application which could be made accessible by mobile devices – e.g. for consultants to remotely log in when being on-site with clients.

The software company would buy IntelliPlan as an off-the-shelf application. As the software is being used over time, it may learn about the peculiarities of the organisation as e.g. the objectives of its different departments. Hence, through using InteliPlan, it becomes increasingly bespoke. The underlying project planning data may become invaluable to the software company and could provide competitive advantage. Apart from the data described in this research, the software could comprise further information such as

- corporate project history
- estimations history (including accuracy of estimations in the past)
- location of reusable components
- further technical expertise (in addition to competencies of resources)

As suggested by an industry expert during the validation of the framework and the software prototype, some functionality of IntelliPlan could be applied to other industries as well, given that corporate strategies are present in a wide variety of corporations outside the field of software. However, this undertaking should be branched out into a separate project, as IntelliPlan would otherwise become too generic, which in turn is a major disadvantage of common project planning tools as shown in the research of this work.

7.6 Future Outlook

Chapter four emphasises that the initiative for positive change in software companies might have to come from software engineers. This entails the danger of creating solutions that remain aimed at technical issues only. However, as shown in this work, for measures to be effective, they may need to consider the entire organisation.

The software engineering discipline will mature over the decades to come, but this process needs to be supported and facilitated. Accreditation of the discipline and consensus on a body of knowledge are still crucial for the discipline to evolve, but these measures need to be complemented by further efforts in the field of commercial software projects. The author believes that considerate project planning described in this research might play a significant role in the prospective evolution of software engineering.

7.7 Concluding Thoughts

The concepts presented in this work can obviously be employed at the discretion of the respective management of software companies. In other words, it could happen that certain principles are overruled, ignored or amended depending on the particular project scenario or corporate situation. This cannot be categorised as a weakness or disadvantage of the Commercial Software Engineering Planning Framework as it simply lies outside the area it can possibly address. However, the framework might be able to significantly mitigate the issue of imprudent interventions, as it is mirrors the strategic impact of amendments to project planning.

With regard to human resource management, Purcell (1989 cited in Armstrong 1992, p. 32) makes the following statement: “If it were possible to demonstrate that ‘enlightened’ or progressive approaches to the management of people were invariably associated with higher productivity, lower unit costs and improved profit, life would be easier for the human resource planner”.
Considering the research findings of this work, similarities of the statement above with software engineering are staggering.

There may be no silver bullet to quickly resolve all issues of the software engineering discipline at once, but these difficulties can and should be seen as a fascinating challenge. It may not often be the case that scholars and practitioners have the chance to significantly contribute to the evolution of the discipline they work in.

Software engineers might be fortunate.

References

Books

Ambler, S.W. & Nalbone, J. & Vizdos, M.J., 2005. *The enterprise unified process, extending the rational unified process.* Prentice Hall PTR.

Armstrong, M. ed., 1992. *Strategies for human resource management, a total business approach.* London: Kogan Page Ltd.

Bainbridge, D.I., 2008. *Introduction to information technology law.* 6th ed. Harlow, Essex: Pearson Education Ltd.

Beck, K., Andres, C., 2004. *Extreme programming explained, embrace change.* 2nd ed. Addison-Wesley.

Bergström, S., Råberg, L., 2003. *Adopting the rational unified process, success with the RUP.* Addison-Wesley.

Boehm, B.W. et al., 2000. *Software cost estimation with COCOMO II.* Upper Saddle River, New Jersey: Prentice Hall PTR.

Bott, F., 2001. *Professional issues in software engineering.* 3rd ed. London: Taylor and Francis.

Brooks, F.P., 1995. *The mythical man-month, essays on software engineering.* Rev. ed. Reading, Massachusetts: Addison-Wesley.

Byars, R., 1997. *Management, skills and application.* 8th ed. Chicago: Irwin.

Callon, J.D., 1996. *Competitive advantage through information technology.* New York: McGraw-Hill.

Currie, D., 2006. *Introduction to human resource management, a guide to personnel in practice.* London: Chartered Institute of Personnel and Development.

Damodaran, A., 2001. *Corporate finance, theory and practice.* 2nd ed. New York: John Wiley & Sons, Inc.

Deek, F.P. & McHugh, J.A.M. & Eljabiri, O.M., 2005. *Strategic software engineering, an interdisciplinary approach.* London: Auerbach.

Earl, M.J. 1989. *Management strategies for information technology.* Hemel Hempstead: Prentice Hall Ltd.

Emam, K.E., 2005. *The ROI from software quality.* Boca Raton, FL: Auerbach Publications

Eva, M., 1992. *SSADM version 4: a user's guide.* McGraw-Hill.

Glass, R.L., 1998. *Software runaways – lessons learned from massive software project failures.* Upper Saddle River, New Jersey: Prentice-Hall Inc.

Ionescu, D. & Cornell, A. eds., 2007. *Real-time systems.* Singapore: World Scientific Publishing Co. Pte. Ltd.

IT Governance Institute, 2007. *IT assurance guide, Using COBIT.* Rolling Meadows, IL: IT Governance Institute.

Johnson, G. & Scholes, K. & Whittington, R., 2008. *Exploring corporate strategy, text & cases.* 8th ed. Harlow, Essex: Pearson Education Ltd.

Khan, R.A. & Mustafa, K. & Ahson, S.I., 2006. *Software quality, concepts and practices.* Oxford: Alpha Science.

Kerr, J., Hunter, R., 1994. *Inside RAD, how to build fully functional computer systems in 90 days or less.* McGraw-Hill, Inc.

Nah, F.F., 2002. *Enterprise resource planning solutions & management.* London: IRM Press.

Porter, M.E., 1980. *Competitive strategy, techniques for analyzing industries and competitors.* 1st ed. New York: Free Press.

Shneiderman, B. & Plaisant, C., 2005. *Designing the user interface.* 4th ed. London: Pearson Education, Inc. / Addison-Wesley.

Sommerville, I., 1993. *Software engineering.* 4th ed. Addison-Wesley.

Tayntor, C.B., 2007. *Six Sigma software development.* 2nd ed. Boca Raton, FL: Auerbach Publications.

Tian, J., 2005. *Software quality engineering, testing, quality assurance, and quantifiable improvement.* Hoboken, N.J.: John Wiley & Sons Inc.

Tunkel, D. & York, S., 2000. *E-commerce, a guide to the law of electronic business.* 2nd ed. London: Butterworths.

Ward, J. & Peppard, J., 2002. *Strategic planning for information systems.* 3rd ed. Chichester, West Sussex: John Wiley & Sons Ltd.

Watson, D. & Head, A., 2007. *Corporate finance, principles & practice.* 4th ed. Harlow, Essex: Pearson Education.

Yourdon, E., 2004. *Death march.* 2nd ed. Upper Saddle River, New Jersey: Prentice Hall Professional Technical Reference.

Conference Papers and Proceedings

Boehm, B.W., 2006. A view of 20th and 21st century software engineering. [online]. In ACM, *International conference on software engineering,* Shanghai, China, May 20–28 2006. Available at: http://portal.acm.org/citation.cfm?id=1134285.1134288 [accessed 2 October 2008].

Bryant, A., 2000. 'It's engineering Jim ... but not as we know it', software engineering – solution to the software crisis, or part of the problem? In The Association for Computing Machinery, *22nd International conference on software engineering: ICSE 2000.* Limerick, Ireland, 4–11 June 2000. ACM: New York.

Dawson, R., 2000. Twenty dirty tricks to train software engineers. In The Association for Computing Machinery, *22nd International conference on software engineering: ICSE 2000.* Limerick, Ireland, 4–11 June 2000. ACM: New York.

Drappa, A. & Ludewig, J., 2000. Simulation in software engineering training. In The Association for Computing Machinery, *22nd International conference on software engineering: ICSE 2000.* Limerick, Ireland, 4–11 June 2000. ACM: New York.

Fraser, S.D., et al., 2007. "No silver bullet reloaded", a retrospective on "essence and accidents of software engineering". [online]. *Conference on object oriented programming systems languages and applications*, Montreal, Quebec, Canada, pp. 1026-1030. Available at: http://portal.acm.org/citation.cfm?id=1297846.1297973 [accessed 1 November 2008]. ACM Press: New York.

Grimson, J.B. & Kugler, H.J., 2000. Software needs engineering – a position paper. In The Association for Computing Machinery, *22nd International conference on software engineering: ICSE 2000.* Limerick, Ireland, 4–11 June 2000. ACM: New York.

Naur, P. & Randell, B., 1969. Software Engineering. *Report on a conference sponsored by the NATO SCIENCE COMMITTEE,* [online], Garmisch, Germany, 7th to 11th October 1968. Available at: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/ [accessed 16 October 2008].

Parnas, D.L., 1994. Software aging. [online]. *16th International conference on software engineering*, Sorrento, Italy, pp. 279-287. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=296790 [accessed 2 October 2008]. IEEE Computer Society Press: Los Alamitos, CA, USA.

Royce, W.W., 1970. Managing the development of large software systems. In *Proceedings of IEEE WESCON.* Agosto, 1970, IEEE. Available at: http://facweb.cs.depaul.edu/jhuang/is553/Royce.pdf [accessed 05 January 2009].

Stewart, G.A. & Cameron, D. & Cowan, G.A. & McCance, G., 2007. Storage and data management in EGEE. [online]. *Fifth Australasian symposium on grid computing and e-Research (AusGrid 2007).* Ballarat, Australia, 30 January – 02 February 2007. Available at: http://portal.acm.org/citation.cfm?id=1274531.1274541 [accessed 21 November 2008].

Internet

Artifact, 2009. *Free, online project management, requirements management & more,* [online]. Available at: http://www.artifactsoftware.com/products/index.html [accessed 8 February 2009].

Buffett, W.E., 1983. *Chairman's letter, Berkshire Hathaway Inc. (to the stockholders of Berkshire Hathaway Inc.)* [online]. (Updated 29 February 2008) Available at: http://www.berkshirehathaway.com/letters/letters.html [accessed 19 January 2009].

Dijkstra, E.W., 1993. *E.W.Dijkstra archive: there is still a war going on (EWD 1165),* [online]. Available at: http://www.cs.utexas.edu/users/EWD/transcriptions/EWD11xx/EWD1165.html [accessed 19 October 2008].

Franklin, S. (IBM), 2005. *Migrating a J2EE project from IBM Rational Rose to IBM Rational XDE Developer v2003 part 1: introduction,* [online]. Available at: http://www.ibm.com/developerworks/rational/library/4285.html [accessed 11 November 2008].

LiquidPlanner, 2008. *LiquidPlanner: online project management software, collaboration, scheduling,* [online]. Available at: http://www.liquidplanner.com/ [accessed 2 February 2009].

Merriam-Webster Online Dictionary, 2009. *Methodology, definition from the Merriam-Webster online dictionary,* [online]. Available at: http://www.merriam-webster.com/dictionary/methodology [accessed 1 February 2009].

Microsoft, 2007. *What is Microsoft Office SharePoint Server?,* [online]. Available at: http://www.microsoft.com/sharepoint/prodinfo/what.mspx [accessed 1 February 2009].

MindTools, 2009. *Influence maps, uncovering where the power lies in your projects,* [online]. Available at: http://www.mindtools.com/pages/article/newPPM_83.htm [accessed 22 February 2009].

RationalPlan, 2009. *RationalPlan project management software, MultiProject version,* [online]. Available at: http://www.rationalplan.com/multi-project-management-software.php [accessed 2 February 2009].

The Omni Group, 2009. *The Omni Group, OmniPlan,* [online]. Available at: http://www.omnigroup.com/applications/omniplan/ [accessed 2 February 2009].

Zoho, 2009. *Online project management, collaboration, timesheet tracking, Zoho projects,* [online]. Available at: http://projects.zoho.com/jsp/home.jsp [accessed 8 February 2009].

Journal and Newspaper Articles and Transactions

Boehm, B.W., 2008. Making a difference in the software century. *Computer,* [online], 41 (3), pp. 32-38. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4476221 [accessed 20 November 2008].

Boehm, B.W., 1988. A spiral model of software development and enhancement. *Computer,* [online], 21 (5), pp. 61-72. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=59 [accessed 3 October 2008].

Brooks, F.P., 1987. No silver bullet, essence and accidents in software engineering. *Computer,* [online], 20 (4), pp. 10-19. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1663532 [accessed 7 October 2008].

Bourque, P., et al., 2002. Fundamental principles of software engineering – a journey. *Journal of Systems and Software,* [online], 62 (1), pp. 59-70. Available at: http://www.gelog.etsmtl.ca/publications/pdf/978.pdf [accessed 25 November 2008].

Elton, D., 2008. IT projects must appeal to emotions to succeed. *Financial Times insert: digital business,* [online]. 8 October, p. 2, p. 4. Available at: http://media.ft.com/cms/f94b585c-9451-11dd-953e-000077b07658.pdf [accessed 24 November 2008].

Glass, R.L., 1994. The software-research crisis. *IEEE Software* [online]. 11 (6), pp. 42-47. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=329400 [accessed 18 November 2008]

Hammer, M., 1990. Reengineering work: don't automate, obliterate. *Harvard Business Review,* 68(4), pp. 104-112. Available at: http://www.city.academic.gr/material/academic_staff/business_administration/morgan/rmn/SHARED/Articles/hammer.pdf [accessed 5 February 2009].

Jones, C., 1995. Legal status of software engineering. *Computer,* [online]. 28 (5), pp. 98-99. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=384135 [accessed 4 November 2008].

Parnas, D.L., 1999. Software engineering programs are not computer science programs. *IEEE Software,* [online]. 16 (6), pp. 19-30. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=805469 [accessed 17 November 2008].

Speed, J.R., 1999. What do you mean I can't call myself a software engineer? *IEEE Software,* [online]. 16 (6), pp. 45-50. Available at: http://portal.acm.org/citation.cfm?id=624634.626062 [accessed 17 September 2008].

van Genuchten, M., 1991. Why is software late?, an empirical study of reasons for delay in software development. *IEEE Transactions on software engineering,* [online]. 17 (6), pp. 582-590. Available at: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=87283 [accessed 24 October 2009].

Research Papers

Abran, A., et al. eds., 2004. *Guide to the software engineering body of knowledge, SWEBOK.* [online]. Available at: http://www.swebok.org/ [accessed 17 October 2008].

Wideman, R.M., 2002. *Comparing PRINCE2® with PMBoK®,* [online]. Available at: http://www.adaptiveframeworks.com.au/prince2/P2vPMBOK.pdf [accessed 9 February 2009].

Bibliography

Allen, T.T., 2006. *Introduction to engineering statistics and Six Sigma, statistical quality control and design of experiments and systems.* London: Springer-Verlag Ltd.

Bratko, I., 2001. *Prolog, programming for artificial intelligence.* 3rd ed. Harlow, Essex: Pearson Education Ltd.

Bugzilla, 2009. *Bugzilla,* [online]. Available at: http://www.bugzilla.org/ [accessed 2 February 2009].

Dijkstra, E.W., 1968. Letters to the editor: go to statement considered harmful. *Communications of the ACM* [online]. 11 (3), pp. 147-148. Available at: http://portal.acm.org/citation.cfm?id=362929.362947 [accessed 25 November 2008].

Edgewall, 2009. *The Trac project,* [online]. Available at: http://trac.edgewall.org/ [accessed 2 February 2009].

Gamma, E. & Helm, R. & Johnson, R. & Vlissides, J., 1995. *Design patterns, elements of reusable object-oriented software.* Reading, Massachusetts: Pearson Education, Inc. / Addison-Wesley.

Pirsing, R.M., 1974. *Zen and the art of motorcycle maintenance.* London: Vintage.

Abbreviations

3D	Three-Dimensional
AI	Artificial Intelligence
BCG	Boston Consulting Group
BCS	British Computer Society
BI	Business Intelligence
CAD	Computer Aided Design
CERN	Conseil Européen (pour la) Recherche Nucléaire
COBIT	Control Objectives for Information and related Technology
COCOMO II	COnstructive COst Model
CRM	Customer Relationship Management
CS	Computer Science
e-Business	Electronic Business
EC	Electronic Commerce
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
HRM	Human Resource Management
ICBF	Implicit Corporate Business Factors
IDE	Integrated Development Environment
IT	Information Technology
ICT	Information and Communication Technologies
MOSS	Microsoft Office SharePoint Server
MS	Microsoft
MVC	Model-View-Controller
NASA	National Aeronautics and Space Administration
NATO	North Atlantic Treaty Organization
NB	Nota Bene
n.d.	no date
OS	Operating System
PESTEL	Political, Economic, Social, Technological, Environmental and Legal
PLM	Product Lifecycle Management
PM	Project Management
PMBOK	Project Management Body Of Knowledge
PRINCE2	Projects IN Controlled Environments II
PSP	Programming Systems Product
RUP	Rational Unified Process
R&D	Research and Development
RAD	Rapid Application Development
ROI	Return on Investment
SCM	Supply Chain Management

SE	Software Engineering
SME	Small and Medium Enterprises
SSADM	Structured Systems Analysis and Design Method
STL	Standard Template Library
SWE	Software Engineering (abbreviation in Figure 7)
SWEBOK	Software Engineering Body Of Knowledge
U.S.	United States
UK	United Kingdom
UML	Unified Modeling Language
USA	United States of America
XML	Extensible Markup Language
XP	Extreme Programming

Appendix A: Counterforce Decision Table

Origin	Force	Potential Reasoning	Counterforce
Senior Management / Corporate Finance	- Not enough time to train for new technologies used in the project - Unrealistic amendments to the project plan: e.g. not enough resources/expertise, budget or development time	- Maximisation of revenues through low cost development - Demands for high quality	- Usage of strategic tools to emphasise long-term benefits for the organisation for strategically important products
Customers	- Unrealistic demands in requirements (e.g. functionality not listed in the specification) - Unrealistic demands for delivery & deployment dates (e.g. reduced quality assurance) - Exerting pressure on Senior Management of the software company	- Demand for high quality in requirement fulfilment & low error rate - Demand for low cost - Fast development, deployment and delivery within deadlines	- Strategy assessment and comparison with the potential of the product -> long-term ROI opportunities for the customer?
Software Engineering / Technical	- Overly optimistic cost/time estimations - Too pessimistic estimations. - Implementation of functionality not defined in the specification: “experimenting around” - Motivational issues with repetitive routine tasks	- Aim for technical brilliance - Avoiding prolonged discussions to justify estimations - Desire to integrate creative and innovative ideas	- Usage of strategic tools to justify relevance of project (e.g. reduced perfectionism for low potential products)
Legal / HRM / Marketing	- Marketing: demand for projects which are not in alignment with the corporate strategy of the company or the customer - HRM: danger of intervention in critical projects to push for rigid implementation of HRM rules - Legal: demand for rigid compliance impacting creativity, inventiveness and motivation to innovate with technicians	- Demands for fair working conditions - Demand for innovative products for promotion purposes - Demand for compliance with laws and policies	- Usage of project-specific factors to point out necessities for the particular project(e.g. additional training for implementing strategically important products)

Table 2 – Counterforce Decision Table

Appendix B: Use Case Diagram for *IntelliPlan*

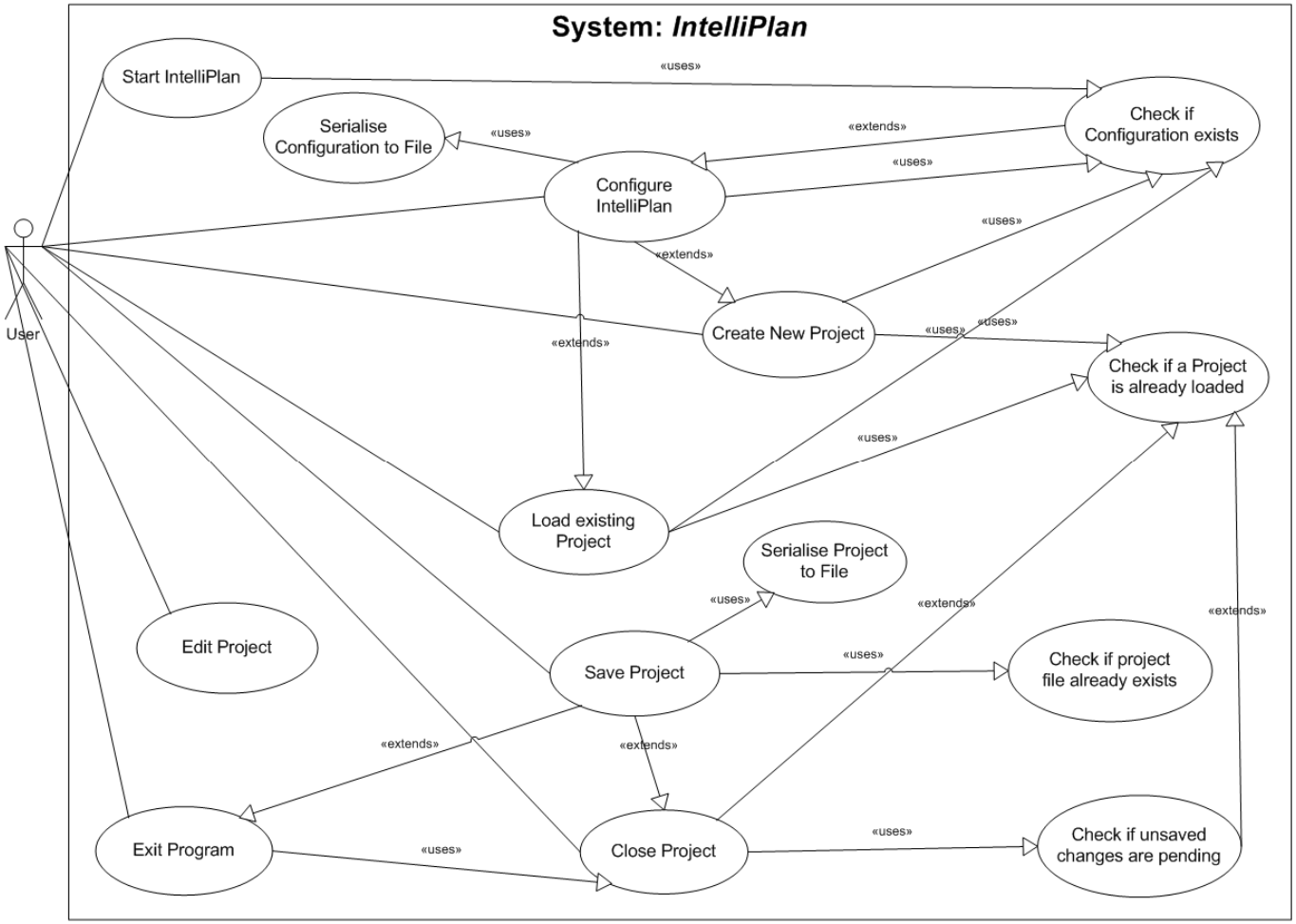

Figure 54 – Use case diagram for IntelliPlan

Appendix C: Class Diagram for *IntelliPlan*

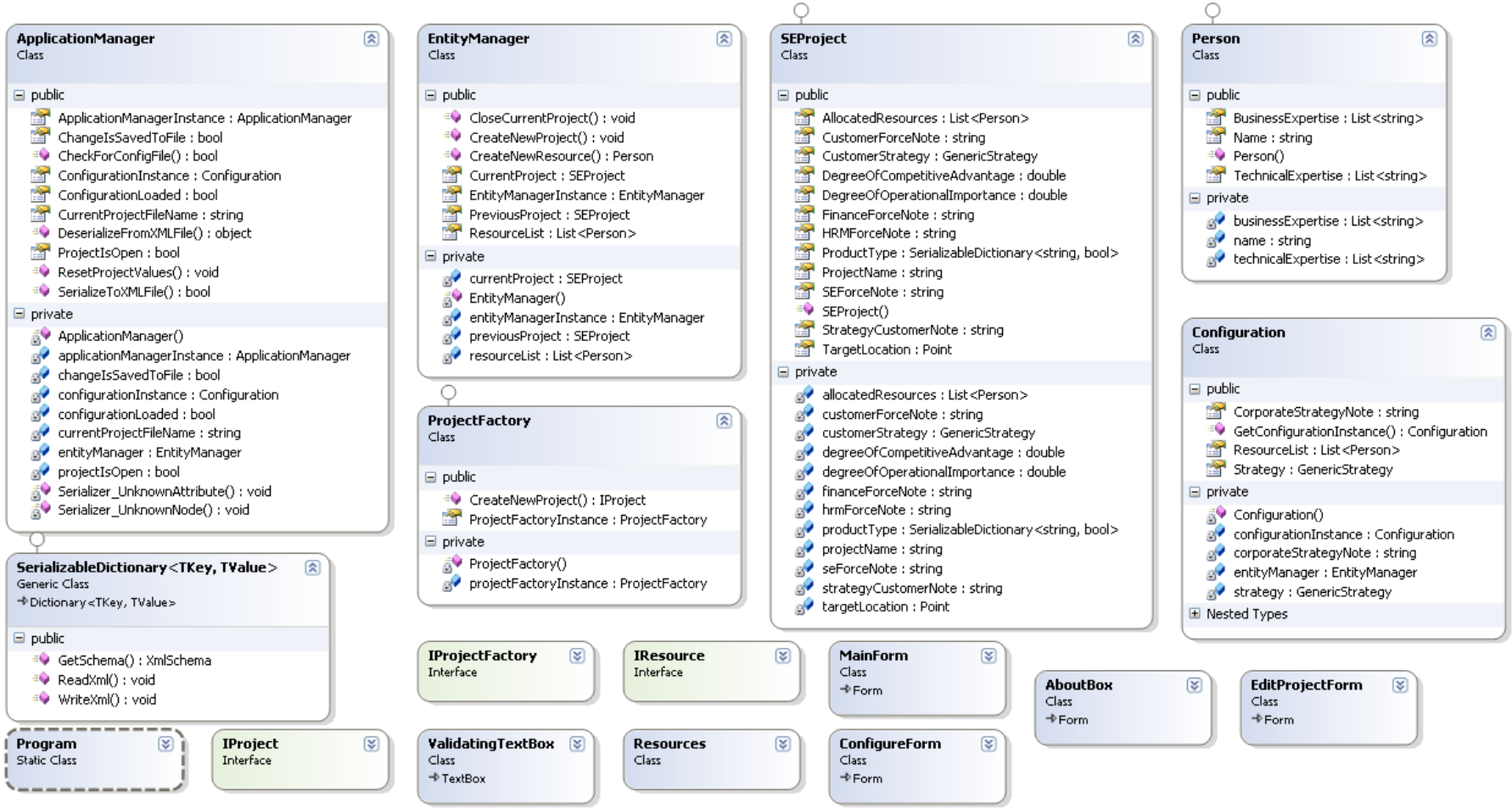

Figure 55 – Class diagram for IntelliPlan

Appendix D: Ethical Consent for Validation

The purpose of this document is to inform the participant about ethical implications concerning the validation of the MSc Dissertation project. Two hardcopies are to be filled in: one is to be retained confidentially by the student and one is to be handed out to the participant.

Validation Details

Date:

Location: Manchester, UK

Professional status of participant: ☐ Industry expert

☐ Academic expert

---

1. The participant is to be informed about the main procedures. Have you fully understood the validation procedures?

   ☐ Yes ☐ No

2. Your participation is voluntary. Please state if you agree to participate in this validation.

   ☐ Yes ☐ No

3. This validation is not observational, but will rather be conducted in a cooperative manner.

4. You may withdraw from this validation of research at any time and for any reason.

5. You may omit questions you do not wish to answer during the interview.

6. Your data will be treated with full confidentiality and, if published, your data will not be identifiable as yours.

7. If you wish, you will have the opportunity to be debriefed about the outcome of the work you participated in (approximately from summer 2009).

---------- ----------

Signature of student Signature of participant

Appendix E: Simulated Project Scenario

QualityProductions is a middle-sized software company which employs 22 people. The organisation consists of the following departments:

- Business Management (3 people), Corporate Finance & Accounting (1 person)
- Technical Development (Software) (12 people)
- Marketing (3 people), HRM (2 people), Legal (1 person)

The company produces freeware tools and low-cost downloadable shareware. Some of these products have the potential to become an off-the-shelf product for the mass-market. One developer is working on these tools full-time.

One of the tools is named *Quantify* and has emerged into a data mining application which is being sold to customers throughout the UK. The software is developed further in ongoing research and development conducted by a team of two developers in full time assignments.

The main proportions of revenues (75%) for the organisation originate from individual software development and maintenance of bespoke software systems. *QualityProductions* is dependent upon two major clients for whom it fixes bugs and implements change requests (development of new functionality). 9 developers are working full time to serve the requirements of the two customers.

Recently, one of the major clients decided to assign a project to *QualityProductions* concerning the development of a new bespoke back-end software system. To complete this project, the management of *QualityProductions* is considering the hiring of two more developers. Furthermore, it wants to distribute the remaining workload across the 9 developers who are familiar with the business area of the client. Also, the business management department plans to withdraw the two resources allocated to the development of *Quantify* for a period of four to six months with the option to continue the work on *Quantify* in part-time assignments.

Appendix F: Contents of the CD

The CD contains an electronic version of this report in two different file formats (MS Word document and PDF) as well as the software prototype IntelliPlan: the disc contains a solution file for MS Visual Studio 2008 and a compiled binary version of the application in form of an installation routine.
The application prototype IntelliPlan requires Microsoft .NET 3.5 to be installed.