DISSERTATIONS IN
FORESTRY AND
NATURAL SCIENCES

Topi Haapio

Improving Effort Management in Software Development Projects

Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences

UNIVERSITY OF
EASTERN FINLAND

TOPI HAAPIO

Improving Effort Management in Software Development Projects

Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences
No 39

Academic Dissertation
To be presented by permission of the Faculty of Science and Forestry for public examination in the Auditorium in Tietoteknia Building at the University of Eastern Finland, Kuopio, on September, 2, 2011, at 12 o'clock noon

School of Computing

Kopijyvä
Kuopio, 2011
Editors: Prof. Pertti Pasanen
Dr. Sinikka Parkkinen, Prof. Kai-Erik Peiponen

Distribution:
Eastern Finland University Library / Sales of publications
P.O. Box 107, FI-80101 Joensuu Finland
tel. +358-50-3058396
http://www.uef.fi/kirjasto

ISBN: 978-952-61-0493-5 (Print)
ISSNL: 1798-5668
ISSN: 1798-5668

ISBN: 978-952-61-0494-2 (PDF)
ISSNL: 1798-5668
ISSN: 1798-5676

Author's address: Tieto Finland Oy
P.O. Box 1199
FI-70210 KUOPIO
FINLAND
email: topi.haapio@tieto.com

Supervisors: University lecturer Anne Eerola, Ph.D.
University of Eastern Finland
School of Computing
P.O.Box 1627
70211 KUOPIO
FINLAND
email: anne.eerola@uef.fi

CTO Olli Lötjönen, Lic.Tech.
Tieto Finland Oy
P.O. Box 1199
FI-70210 KUOPIO
FINLAND
email: olli.lotjonen@tieto.com

Reviewers: Associate Professor Emilia Mendes, Ph.D.
The University of Auckland
Computer Science department
Private Bag 92019
AUCKLAND
NEW ZEALAND
email: emilia@cs.auckland.ac.nz

Professor Tommi Mikkonen, D.Tech.
Tampere University of Technology
Department of Software Systems
P.O.Box 527
FI-33101 TAMPERE
FINLAND
email: tommi.mikkonen@tut.fi

Opponent: Professor Matti Rossi, Ph.D. (Econ)
Aalto University
School of Economics
P.O.Box 21210
FI-00076 AALTO
FINLAND
email: matti.rossi@aalto.fi

ABSTRACT

A software supplier organization strives to estimate the effort needed in building software as accurately as possible to ensure the project's budget and schedule, and the success of resource allocation. Despite the numerous effort estimation approaches and applications available, the estimates have remained inaccurate. The objective of this thesis is to improve the management practices of software development project effort, resulting in increased effort estimate accuracy.

In the quest of its goal, the thesis commences by presenting the theoretical background and the key concepts related to software project effort management, followed with a description of the iterative research approach. The research problems are formulated based on the organizational problems acknowledged in the software engineering literature and the ones observed at the research site, Tieto Finland Oy. To address the research problems, research artifacts are built and evaluated with the constructive research methodology.

The results are presented in five research papers. The main research results include frameworks for both improving and managing software project effort, a defined new set of project activities and their significance in terms of project effort, recommendations for adopting new project activities in an efficient manner, a process and method for effort assessments, and findings from a software project effort data mining experiment.

This thesis focuses on an activity set of general software project activities, referred as the non-construction activities, which are found to be one of the major software project activity categories besides software construction and project management. This finding is complemented with recommendations for efficient adoption of new project activities and set of activities. By improving the adoption mechanisms of new project activities it is more likely that effort is registered on correct activities, which ensures reliable effort input for effort

assessments. The assessment results are improved with the stepwise software project effort assessment method introduced in this thesis. When applied, the research results aim to improve the quality of effort data, which can then be utilized in the effort estimation method calibration in order to achieve more accurate estimates.

The research findings and constructed artifacts are beneficial for project managers and effort analysts who can better manage their effort-related project activities through-out project's lifecycle.

Universal Decimal Classification: 004.41; 005.8

Inspect Thesaurus: project management; software development management; software engineering; software process improvement; software cost estimation; software metrics

Acknowledgements

This work was carried out at Tieto Finland Oy for the School of Computing at the University of Eastern Finland (former University of Kuopio) during 2003-2011. The initial idea of the role of the non-construction activities on effort estimation came from Seppo Hartikka at Tieto Quality. I thank him for raising this issue up.

I thank the supervisors of this thesis, Dr. Anne Eerola (University of Eastern Finland) and CTO Olli Lötjönen (Tieto Finland Oy) for their invaluable guidance and comments both for this thesis and the research papers involved.

Associate Professor Emilia Mendes (The University of Auckland) and Professor Tommi Mikkonen (Tampere University of Technology) kindly accepted the role of a reviewer. I wish to thank for their efforts and feedback on my work.

I thank all my friends in academia for supporting and encouraging me in my research. I want to especially thank Dr. Tim Menzies (West Virginia University) for our research co-operation in Morgantown, Dr. Raimo Rask (University of Eastern Finland) and Dr. Uolevi Nikula (Lappeenranta University of Technology) for their thorough feedback on my Ph.Lic. thesis. The feedback was invaluable in improving this thesis also, Professor Jarmo J. Ahonen (University of Eastern Finland) who enhanced my research into the software process improvement, Dr. Seppo Lammi (University of Eastern Finland) from my supervising team for his uplifting and positive attitude, and Mr. V. Michael Paganuzzi, MA, (University of Eastern Finland) for his linguistic advice on this summary and research papers.

I thank my managers at Tieto Corporation for letting me spend time on conducting research during working hours. I thank Tieto Corporation and the University of Eastern Finland

for funding my participation to conferences. I also thank the support of the KYTKY program (EU funded development program for research co-operation between the software industry and the University of Kuopio), and Nokia Oyj for the Nokia Scholarship, which made my visit at the West Virginia University possible.

My personal thanks go to all my friends and relatives who have supported me in a way or another during these years and my research. I especially wish to thank my mother Arja, my sister Anu and her family, and my wife Minna's family. Mrs Terttu Jalkanen is always there when needed - without her I would have missed a few deadlines!

My deepest gratitude and love go to my wife Minna who showed remarkable understanding for the long days this research consumed. I also like to thank her professionally on commenting my work.

Kuopio, August 2011 *Topi Haapio*

LIST OF ABBREVIATIONS

3GL	3rd Generation Language
4GL	4th Generation Language
BI	business intelligence
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
COCOMO	Constructive Cost Model
DSI	delivered source instructions
EAF	Effort Adjustment Factor
ELOC	effective lines of code
FP	function point
FPA	Function Point Analysis
FSS	Feature Subset Selection
GQM	Goal/Question/Metric
IT	Information Technology
KDSI	kilo delivered source instructions
KLOC	kilo lines of code
LOC	lines of code
MMRE	Mean Magnitude of Relative Error
MRE	Magnitude of Relative Error
NLOC	non-commented lines of code
PRED(*l*)	Prediction at Level *l*
QM	quality management
RE	Relative Error
RP	research problem
RUP	Rational Unified Process
SD	standard deviation
SDM	Software Development Management (framework)
SPI	software process improvement
SPICE	Software Process Improvement and Capability dEtermination
TQM	Total Quality Management
UCP	Use Case Points
WBS	work breakdown structure

LIST OF ORIGINAL PUBLICATIONS

This thesis is based on the following research publications, which are referred to in the text by their Roman numerals I - V:

I Haapio, T., 2007. A Framework for Improving Effort Management in Software Projects. *Software Process: Improvement and Practice*. 12(6), pp. 549-558.

II Haapio, T. & Menzies, T., 2011. Exploring the Effort of General Software Project Activities with Data Mining. *International Journal of Software Engineering and Knowledge Engineering*. In press, accepted 29 March 2011.

III Haapio, T. & Ahonen, J.J., 2006. A Case Study on the Success of Introducing General Non-construction Activities for Project Management and Planning Improvement. In: Lecture Notes in Computer Science 4034, J. Münch and M. Vierimaa (eds.), *7th International Conference on Product-Focused Software Process Improvement (PROFES 2006)*. Amsterdam, The Netherlands, 12-14 June 2006. Berlin Heidelberg: Springer-Verlag, pp. 151-165.

IV Haapio, T. & Eerola, A., 2010. Software Project Effort Assessment. *Journal of Software Maintenance and Evolution: Research and Practice*. 22(8), pp. 629-652.

V Haapio, T., 2007. A Framework for Effort Management in Software Projects. In: B. Kitchenham, P. Brereton, and M. Turner (eds.), *11th International Conference on Evaluation and Assessment in Software Engineering (EASE 2007)*. Keele, UK, 2-3 April 2007. The British Computer Society eWiC, pp. 73-82.

AUTHOR'S CONTRIBUTIONS

I Sole work of Topi Haapio.

II Topi Haapio collected and analyzed the data. Tim Menzies suggested the learners for the data mining experiment, and assisted in interpreting the results. Topi Haapio provided the implications to the practice, and Tim Menzies to research. The paper was written by Topi Haapio, and commented and enhanced by Tim Menzies.

III Topi Haapio set up and conducted the survey, and analyzed the survey results. The paper was written by Topi Haapio, and commented and edited by Jarmo J. Ahonen.

IV Topi Haapio developed the effort assessment process and method, which were improved together with Anne Eerola. The paper was written by Topi Haapio, and commented by Anne Eerola.

V Sole work of Topi Haapio.

Contents

1 Introduction

Accurate effort estimation is a crucial task for software business progression: for customers, acquiring software products or making project contracts for tailored software implementation, accurate effort estimation enables adherence to schedule and budget without delay in deployment and introduction (Conte, et al., 1986; Sommerville, 2001). Software providers require cost, price, and time-to-market calculations which are not possible without knowledge of effort and its distribution. Workload estimation is needed for work and staffing plans. Sufficiently accurate effort estimation is important for software engineers, because successful resource allocation decreases working pressure and haste.

Effort is, however, frequently underestimated (Gruschke and Jørgensen, 2008; Kitchenham, et al., 2009). During our research, we have identified several shortcomings related to both estimating and managing the effort which can explain poor estimates. We have identified these gaps by reviewing software engineering literature of past two decades and by observing the practices in the software industry. In the following, we present these gaps in concise, and return to them in more detail in sub-chapter 3.1, with our aims to fill these gaps.

The numerous formal models, methods and tools available for estimation have reached accuracy levels which are too low for the software industry companies. In practice, the same level of accuracy can be achieved with the informal expert judgment technique, which has remained as the most commonly employed estimation technique (Jørgensen, 2007). Two factors are emphasized in respect to the difficulty in producing accurate estimates: software sizing and the available data for estimations (Armour, 2002). Moreover, it has been argued that the formal effort estimation models are too complex and uncertain for

practical use (Sommerville, 2001). The applications of these models are usually not transparent, i.e., the factor weights used for effort derivation are not obtainable in order to validate them. Transparency would be required by the estimator to evaluate the reasonability of the gained estimate, i.e., what comprises the effort and the costs.

An important, yet overlooked, factor explaining poor estimates is the light consideration on different activities involved with the project. The estimators employing the expert judgment technique frequently consider only those activities that they are in relation with in a software project and focus on the effort of actual software construction (Jørgensen, 2004b, 2005; Jørgensen and Sjøberg, 2004). Moreover, the focus in effort estimation research and estimation applications has been on software construction (i.e., analysis, design, implementation and testing) and project management (Boehm, et al., 2000; MacDonell and Shepperd, 2003), whilst neglecting other software project activities. Software engineering literature (e.g., (Pressman, 2005; Royce, 1998; Sommerville, 2001)) focuses on project management and effort concepts, but the emphasis is on effort estimation and effort-related planning (e.g., scheduling) rather than on the total management of effort. Effort has not been seen as an independent area of management like risk or quality. Literature discusses risk management, quality management and configuration management individually but effort is covered as a part of software project management. In other words, research has particularly focused on improving effort estimation, and has concentrated on the software construction effort of the software projects.

Another reason for the effort estimate inaccuracy can be the inadequacy of previous projects' effort data collection when effort is both reported badly and collected without analyzing it properly afterward. The collected effort data is used for both improving team performance in upcoming project phases and project and in calibrating the weights that adjust the factors and drivers which are used for the effort estimate derivation in the organization in question. The proposed processes for post-

mortem analysis describe post-mortem of the whole project, and provide general guidelines without a detailed method to analyze effort. Furthermore, the proposed analysis processes require rather large resources (both time and personnel) which, in practice, are quite limited in the software industry.

In spite of the vast research on effort and cost estimation, the estimations have remained inaccurate. Besides effort estimation, a software project involves various other effort-related functions throughout the project's lifecycle (Figure 1). These effort-related functions influence on effort estimation. Some effort-related functions exist also before and after the project. Therefore, we propose a broader perspective on the topic.

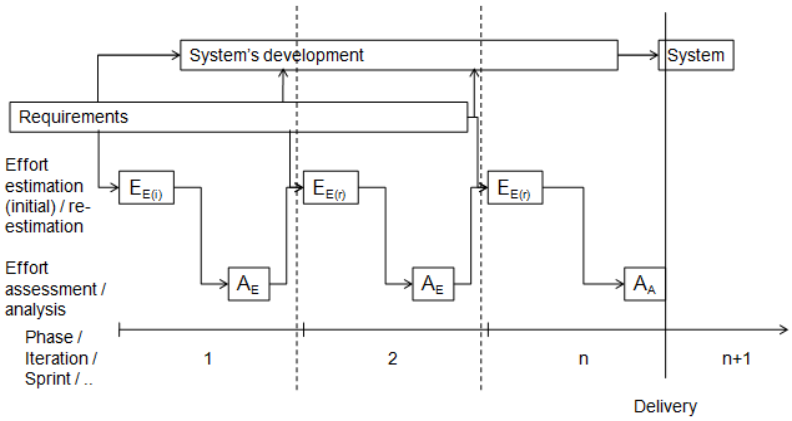

Figure 1. *The system's development and consideration of effort in software projects*

This thesis considers effort-related software project functions in a more complete manner, and introduces improvement results to reduce effort estimation inaccuracy. The results are applicable in the software industry, particularly in the custom software development projects. The results are to be applied together but can also be applied independently. The effort-related functions are parts of software project's effort management, which is presented in this thesis in a form of a framework.

The thesis consists of five original publications and a summary reviewing the publications. The organization of the

summary is as follows. Chapter 2 presents a theoretical background of the key concepts related to effort management. Chapter 3 presents the research approach with the key research problems, and the employed methodologies.

Chapter 4 takes an overview on the research publications. The chapter begins with describing the relation of the research papers to each other, followed with the summaries of five research papers: after describing a research framework for effort management applied in this thesis, four improvement proposals are introduced concerning several software development project phases and their effort-related functions. Chapter 4 ends with an evaluation of the research papers and results, and a discussion on the contributions this thesis offers.

Conclusions with suggestions for further research are presented in Chapter 5.

The Appendices include the quantitative effort data of the 32 software projects examined, and presents further literature for the non-parametric effort and cost estimation techniques.

2 Theoretical background

This chapter introduces the key concepts related to this thesis, and the theoretical background for the key activities related in managing effort in a software project; effort estimation, effort collection, and effort assessment.

2.1 KEY CONCEPTS

The imprecise estimation terminology used both in practice and in software engineering literature is not only a confusion in the literature but also, argued by Grimstad, et al., (2005), a contributing factor in frequent software project cost overruns, since the interoperability and communication of the estimates become difficult. Although Grimstad, et al., (2005) refer in their study to the term variations of the effort estimate, the concepts of effort and cost estimations are also frequently mixed, and are used often faulty as synonyms. Therefore, we first give special consideration for the definitions of effort, estimate, and effort and cost estimations related to software projects. This thesis aims to improve the estimation of effort. Second, the concept of effort distribution is defined. Effort distribution is one of the key concepts for our perspective into effort management. Third, closely related to effort distribution, the concept of work breakdown structures is defined, since effort estimation is in many cases based on these structures, or effort is managed with them. These concepts are parts of effort management, which we define as the fourth concept. Fifth, the different project types are defined, since the case studies of this study concern one of them, custom software development projects. Sixth, the software process improvement and related process capability maturity models are presented. Seventh, the concept of evaluation metric is defined and the well-established metrics are presented. We

not only evaluate our research results with these metrics but also utilize them in a method this thesis introduces. Finally, the software size metrics are presented.

2.1.1 Effort, costs, estimate, and effort and cost estimation

The *effort* of a software development project can be generally defined as the time consumed by the project, and it can be expressed as a number of person hours, days, months or years, depending on the size of the project (Chatters, et al., 1999). Brooks (1975) has defined effort as the product of people and time, i.e., effort = people * time. Effort is estimated in most projects, especially in projects employing traditional project management methods, to derive the project costs that are needed to justify the business case. In agile software development, the project costs are derived from the estimated software size (Cohn, 2008). An *estimate* is a probabilistic assessment with a reasonable accurate value of the center of a range. Formally, an estimate is defined as the median of the (unknown) distribution (Fenton and Pfleeger, 1997). An estimate is a prediction; hence, an estimation model can be considered as a prediction system. *Effort estimation* is often used as a synonym for cost estimation but, to be precise, *cost estimation* is derived from the effort estimate by valuing the effort using the cost of project personnel and some other major project costs. Thus the outcome of effort estimation is a time value, whereas the cost estimate outcome is some monetary value (Conte, et al., 1986; Fenton and Pfleeger, 1997; Sommerville, 2001).

2.1.2 Effort distribution

Although effort can be distributed in a project between project activities or project phases in many different ways, *effort distribution* is a less-investigated effort estimation area. Indeed, it has been disputed whether it is useful to distribute effort in the first place (Blackburn, et al., 1996; MacDonell and Shepperd, 2003). Yet examples exist in the software engineering literature; Brooks' (1975) suggestion for rule-of-thumb effort distribution for software construction among the first in the mid-1970s.

Effort distribution can also be used to compare different projects with each other (cf. (Heijstek and Chaudron, 2008)).

2.1.3 Work breakdown structure (WBS)

Effort is distributed on software project activities and sets of activities that form a *work breakdown structure*, WBS. A WBS is a particular defined tree-structure hierarchy of elements that decomposes the project into discrete work tasks (Royce, 1998; Wilson and Sifer, 1988). It can be very useful to organize project activity elements into a hierarchical structure for project budgetary planning and control purposes (Boehm, 1981). An early establishment of a WBS helps to divide the effort into distinct work segments that can be scheduled and prioritized (Agarwal, et al., 2001).

WBS is also required by the currently employed capability maturity models. For example, the staged representation of CMMI (SEI, 2006; SEI, 2010) requires WBS on its maturity level 2. However, no standardized way to create a WBS exists, and the software engineering literature provides only a few general WBS including ones applied with the two COCOMO models (Boehm, 1981; Boehm, et al., 2000), the *Rational Unified Process* (RUP) activity distribution (Royce, 1998), and the ISO/IEC 12207 work breakdown structure for software lifecycle processes (ISO, 1995). It is noted that although the concept and practice of using a WBS is well established, the topic is largely avoided in the published literature because the development of a WBS depends on the project management style, organizational culture, customer preference, financial constraints, and several other project-specific parameters (Royce, 1998).

Another structure for work breakdown is provided in the OPEN process Framework (Henderson-Sellers, 2003), which in one of the five major framework components is Work Units, which results as a Work Product. A *Work Unit* is defined as a functionally cohesive operation performed by a Producer (people). The three major classes of Work Unit are Activity (e.g., 'project planning' or 'modeling and implementation'), Task (e.g., 'code' or 'evaluate quality'), and Technique (e.g., 'class naming',

'prototyping' or 'unit testing'). These classes form pairs and are linked with each other, e.g., Activities are linked with Tasks, and Tasks with Techniques.

2.1.4 Effort management

The concept of effort management is rarely used in software engineering or information system research. The concept was introduced in information system research first in 1995. Young (1995, p. 716) defines *effort management* as a management area that "tracks the commitment of resources against undertakings, both project and non-project". In fact, Young (1995) emphasizes the non-project consideration over project. We, on the other hand, implicitly limit our consideration to software projects, and define effort management as an organized process to estimate, collect, monitor and control, and analyze effort related to software projects and their activities.

Effort management can be considered to include all necessary functions which manage effort in a software project. These functions include effort estimation, effort collection, and effort assessment and analysis, for instance. Simplified, during project planning, effort is initially estimated with the method in use. The estimate is revised during project execution with both collected and assessed effort data from the project. During project closure, the delivered project and its effort are analyzed, and a project final report is drawn.

To assist effort management, some frameworks have been proposed: *Software Development Management* (SDM) framework (Tsoi, 1999), and the frameworks proposed in (Fairley, 1992; Vesterinen, 1998). These frameworks are fairly limited, both in terms of project lifecycle phases and effort functions. For example, the SDM framework is limited mostly to the pre-project and project phases. The proposed frameworks also concentrate mostly on the effort estimation function.

2.1.5 Custom software development project

A *project* holds following major characteristics (Gray and Larson, 2006):

1. an established objective,
2. a defined life span with a beginning and an end,
3. usually, the involvement of several departments and professionals,
4. typically, doing something that has never been done before,
5. specific time, cost, and performance requirements.

Software projects can be divided into five different categories (Pressman, 2005): concept development projects, new application development projects, application enhancement projects, application maintenance projects, and re-engineering projects. The concept development projects are initiated to explore a new business concept or application of a new technology. The new application development projects are undertaken as a consequence of a specific customer request. When the existing software undergoes major modifications that are observable to the end-user, the project type is an application enhancement project, whereas an application maintenance project corrects, adapts, or extends the existing software in ways that may not be visible to the end-user immediately. The re-engineering projects are undertaken with the intent of rebuilding an existing legacy system in whole or partially (Pressman, 2005).

The project type considered in this thesis' case studies is a *custom software development project*, which refers to a new application development project, in which the software is implemented customized. Software projects often implement customized software (Herzum and Sims, 2000; Szyperski, 1999). Custom software is a system that is commissioned by a certain customer for certain special requirements (Sawyer, 2001; Sommerville, 2001). Although custom software is typically implemented from scratch, the use of ready-made components is endeavored (Boehm, et al., 1995; Herzum and Sims, 2000; Sawyer, 2001; Szyperski, 1999). The components can be either

self-made component packages, or applications made by a third party (Herzum and Sims, 2000; Sawyer, 2001; Szyperski, 1999). Custom software is preferred as the customer gets software that meets the set requirements, and software does not contain extra functionality. Moreover, usually the rights for software are transferred to the customer.

The effort estimations of these different project types vary although same approaches or methods can be used for several types. The decisive difference is the amount and depth of the information in hand to make the estimations. If a software application exists at the point of the estimation, the knowledge on the project is naturally higher than in situations, where a new application is constructed. Hence, the estimations on application enhancement or maintenance projects should be more accurate (Boehm, 1981; Boehm, et al., 1995). In fact, the maintenance projects are excluded from this research since maintenance project effort estimation is sufficiently accurate both in the software industry and in the academia. For example, the findings for estimating the maintenance effort of object-oriented systems are promising with low misestimating values (Fioravanti and Nesi, 2001).

In this thesis, we consider aspects that are commonly associated with the traditional software projects since projects applying agile software development approach (Cohn, 2008) emerged at the research site at the end of this research, and because software deliveries still are in many cases carried out with traditional project management methods. In fact, Dybå and Dingsøyr (2008) conclude that instead of abandoning the traditional project management principles in agile deliveries, one should take advantage of them and combine them with agile project management. Also, the evidence suggests that agile methods are not necessarily the best choice for all projects, e.g., large ones (Dybå and Dingsøyr, 2008).

2.1.6 Software process improvement (SPI) and capability maturity models

Effort management can also be the focus of the *software process improvement* (SPI) activity. SPI means understanding the existing processes and improving them to achieve improved product quality and to reduce costs and development time. SPI is usually an activity that is specific to an organization (Sommerville, 2001).

SPI has its origins in the *Total Quality Management* (TQM). The principles of the statistical quality control in the product quality management from the 1930's were further developed in the 1980's with a premise that real process improvement must follow a sequence of steps, i.e., make the process concrete, repeatable and measurable. The premise for improvement is that it is managed which in turn require that it is measurable (Zahran, 1998).

The SPI activity initiative can arise from employing a capability maturity model. The maturity models are used to improve the related processes. The most widely known and employed capability maturity models include ISO/IEC 15504 (formerly known as SPICE) (ISO, 1993), and CMMI (SEI, 2006; SEI, 2010) which evolved from CMM (Paulk, et al., 1993).

The capability maturity models set different effort-related requirements on software projects. However, these requirements are presented in a rather fragmented manner in models' descriptions. For example, CMMI requires elements of the effort management. Mostly, the effort management relates to the project management process area in CMMI. Like CMM, CMMI requires an organization's measurement repository, which is used to collect and make available measurement data on processes, e.g., effort and cost estimates, and the actual effort and costs, to analyze the measurement data (SEI, 2006; SEI, 2010). Moreover, the staged representation requires work to be arranged as work elements and their relationship to each other and to the end product, i.e., into a work breakdown structure. The structure is required to estimate the scope of the project, and to plan the project resources and to manage configurations

(SEI, 2006; SEI, 2010). It is notable, that even though the latest release of the CMMI model is enhanced with guidance for organizations using Agile methods, the process maturity is still measured based on attributes associated with the traditional approach of project management such as effort and work breakdown structures (SEI, 2010).

SPI and the employment of capability maturity models are under enormous interest of research. Numerous studies have been conducted applying capability maturity models in software industry (e.g., (Clarke, 1997; McBride, et al., 2004; Rautiainen, et al., 2002)). However, studies describing improvement activity in software project effort management context are not very common. It is also argued that maturity models provide a good basis for SPI, but also an excessive overhead if deployed in full (Rautiainen, et al., 2002), although the process maturity correlated negatively with project effort, i.e., increment changes in process maturity result in reduced effort (Clarke, 1997). Moreover, it is claimed that SPI do not take enough in consideration that different business situations require different processes (Rautiainen, et al., 2002). Furthermore, it is argued that maturity models may not be the best models to measure maturity in management processes such as effort management, as opposed to the technical processes of developing the software, since models have an underlying process model that view software development activities in an industrial production-like fashion, focusing attention on the flow of work from one process to another (McBride, et al., 2004).

2.1.7 Method evaluation criteria

Both effort estimates and effort estimation methods are evaluated with criteria metrics: established metrics include *Prediction at Level l* (PRED(*l*)), the *Magnitude of Relative Error* (MRE), and the *Mean Magnitude of Relative Error* (MMRE). Besides for their conventional use, we employ criteria metrics as a part of our effort assessment method itself (Paper IV).

The *Magnitude of Relative Error* (Conte, et al., 1986; Fenton and Pfleeger, 1997), MRE, is derived from *Relative Error* (RE), and determined as follows:

$$MRE = \left| \frac{E_E - E_A}{E_A} \right| \qquad (1)$$

where E_E = estimated effort and E_A = actual effort.

MRE considers both types of errors, overestimates and underestimates (Kemerer, 1987), which are equally unfavorable since complete effort accuracy is pursued. The closer to zero the MRE value is, the better the prediction (Conte, et al., 1986; Kemerer, 1987). In the light of some studies it seems that effort estimation accuracy in software projects in terms of MRE follow normal distribution (Milicic and Wohlin, 2004). As the project proceeds and the effort is re-estimated, E_E closes on E_A and the MRE value gets smaller, because estimates get more accurate as more information is available (Boehm, 1981; Boehm, et al., 1995). Usually an organization sets a certain limit MRE value to project effort estimations and, if it is exceeded, the reasons are explored and some actions taken to prevent further excesses.

Prediction at Level l (PRED(*l*)) describes how many projects in a set of projects have a MRE ≤ *l* (Conte, et al., 1986; Fenton and Pfleeger, 1997):

$$PRED(l) = \frac{k}{n} \qquad (2)$$

where k = the number of a set of n projects whose mean magnitude of relative error is less than equal to l.

For example, PRED(0.25) is the percentage of estimations that fall within 25% of the actual value (Shepperd and Schofield, 1997). Hence, PRED(*l*) identifies those estimates that are generally accurate (Menzies, et al., 2005; Shepperd and Schofield, 1997). Since Conte et al., (1986) set l to 0.25 it has been established as one of the most widely used evaluation criteria in effort estimation research (e.g., (Boehm, et al., 1995; Gray, et al., 1999; Menzies, et al., 2005; Shepperd and Schofield, 1997; Shepperd, et al., 1996) together with MRE. However, as the choice of the accuracy measure depends on the objectives of the stakeholders (Shepperd and Schofield, 1997), the MRE value of

0.25 is considered in many cases too poor and beyond practical use for the software industry, although values as high as 0.35 have also been employed (Jalote, 2000). The accepted MRE value for project effort estimations in organizations is usually set by the business and quality responsible.

Another well-known metric is the *Mean Magnitude of Relative Error* (MMRE):

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i \qquad (3)$$

where n is a set of projects and MRE is Magnitude of Relative Error (Conte, et al., 1986).

2.1.8 Software size metrics

A common input variable for an effort and cost estimation model or method is the size of the constructed software. A software product is a physical entity, thus it can be described in terms of its size (Fenton and Pfleeger, 1997). Software size is suggested to be described with three fundamental attributes (Fenton and Pfleeger, 1997): length, functionality, and complexity. Software size is usually measured either with a size-related or a function-related measure. The size-related measures concern the size of some output from an activity. The most common size-related measure is lines of delivered source code (*lines of code*, LOC, which is typically presented as *kilo-LOC*, KLOC) (Sommerville, 2001). However, there is dispute on the definition of LOC. The most widely accepted definition for LOC is a non-commented source statement, i.e., any statement in the program except for comments and blank lines (Fenton and Pfleeger, 1997).

Other examples of well-known length-related measures include the variants of LOC (e.g., *non-commented lines of code* NCLOC or *effective lines of code* ELOC), *delivered source instructions* (DSI), and the complex measure of Halstead's software science (Conte, et al., 1986; Fenton and Pfleeger, 1997). These measures were suggested initially for the procedural programming languages. More suitable length-measures for

object-oriented programming consider the number of objects, classes, or methods, for instance (Fenton and Pfleeger, 1997).

The function-related measures are related to the overall functionality of the delivered software, and they are independent from the programming paradigm. The best known function-related measures are function points and object points (Sommerville, 2001). *Function points* are intended to measure the amount of functionality in a system as described by a specification (Fenton and Pfleeger, 1997). *Object points* are an alternative to function points when 4GLs or comparable languages are used for software development (Sommerville, 2001). Object point approach, first introduced in (Banker, et al., 1991), is a synthesis of a procedure and reported productivity data (Banker, et al., 1994). They are also used in the COCOMO II estimation model for early size measuring (Fenton and Pfleeger, 1997; Sommerville, 2001). The object points are easier to estimate than function points from a high-level software specification. The number of object points in a program is a three-level weighted estimate of the number of separate displayed screens, the number of produced reports and the number of 3GL modules that must be developed to supplement the 4GL code (Fenton and Pfleeger, 1997; Sommerville, 2001).

Both function points and object points can be used to early stage estimation. Estimates of these parameters can be made as soon as the external interactions of the system have been designed. At this stage, it is very difficult to produce an accurate LOC estimation. Early estimates are essential when using the algorithmic cost estimation models (Sommerville, 2001). Moreover, function points are language-independent so productivity in different programming languages can be compared (Sommerville, 2001). Hence, several effort and effort-related measures can be counted in effort assessments in relation to *function points* (FP) such as productivity (e.g., hours per FP), quality (e.g., defect removal efficiency), and finance (e.g., cost per FP, repair cost ratio) (Garmus and Herron, 2001).

Feature Points, an adaptation to function points, were tailored to size estimation for computationally intensive systems such as

real-time systems (Fairley, 1992). In the Feature Point approach, the number of program's algorithms is added as the sixth function point factor. Moreover, some of the function point weighting factors changed in Feature Points, and the fourteen complexity factors were reduced down to two: logic complexity and data complexity.

Besides function points, object points and feature points, functional sizing approaches include use case based estimation (Pressman, 2005). This technique, however, involves several uncertainties, for example how the use cases are presented. The use case points are calculated based on the number and complexity of the use cases, their scenarios and different actors, and weights related to the technical and environmental factors. An effort estimation method based on the use case estimation technique, *Use Case Points*, was introduced by Karner in 1993 (Mohagheghi, et al., 2005), and has been incorporated into RUP (Royce, 1998).

Another sizing approach is *story points*, which is applied in agile estimation (Cohn, 2008). In the story point technique, the development team values user stories with the relative size of story points based on the analogy of previously implemented stories. An agile estimation technique based on story points, *planning poker*, was introduced by Grenning in 2002 (Cohn, 2008). Planning poker combines expert judgment, analogy, and disaggregation of the user stories into smaller entities. During iterations to reach consensus on the final estimate for an entity, the team members state their story point size value at the same time with cards of a non-linear scale, and the differences between the indicated values are then discussed before next iteration. Based on the team's known rate of progress, *velocity*, the duration of the project can be derived from the combined final relative size estimates.

Out of several approaches into software sizing two approaches and four metrics are employed in practice: size in terms of program code (KLOC) and program functionality (function points, use case points, and story points). Since program functionality is known before the number of

implemented program code lines, function-based approach seems to be more suitable for effort estimation purposes, in practice.

2.2 KEY FUNCTIONS IN MANAGING SOFTWARE PROJECT EFFORT

This sub-chapter introduces key functions related to managing effort in a software project; effort estimation, effort collection, and effort assessment.

2.2.1 Effort estimations and re-estimations

The effort of a software project can be estimated and modeled, and estimation techniques can be classified in several ways. In the following, we present the evolution from the first effort and cost estimation approach classifications by Boehm (1981) and Conte, et al., (1986) to the later classifications by Fairley (1992), Fenton and Pfleeger (1997) and Briand, et al., (1999, 2000).

Boehm (1981) divided the methods into seven categories: algorithmic models, expert judgment, analogy, Parkinson, price-to-win, top-down, and bottom-up. The algorithmic models include linear models, multiplication models, analytic models, tabular models, and composite models. Later, this division was reduced to two main antithesic approaches: algorithmic and non-algorithmic approaches.

Conte, et al., (1986) divided the approaches of effort estimation on the highest level into micro-model and macro-models of effort. The macro-models of effort were divided further into historical-experiental models, statistically-based models, theoretically-based models, and composite models. The historical-experiental models include expert judgment technique. The statistically-based models include techniques such as regression analysis, linear models, and non-linear models.

In 1992, Fairley (1992) addressed the different estimation techniques which included the traditional approaches

(categorized into empirical techniques, regression techniques, and theory-based techniques) and the advances in the popular approaches at that time (advances in analogy-based estimation, function point techniques, regression modeling, theory-based models, and in size estimation). Fairley (1992) predicted that analogy-based methods with expert system decision rules will emerge into future estimations.

Fenton and Pfleeger's (1997) division was based much on Boehm's division, but included only four categories: expert opinion, analogy, decomposition, and models. Each of these techniques can be applied either bottom-up or top-down.

Briand, et al., (1999, 2000) divided the effort estimation techniques into parametric and more advanced non-parametric modeling techniques (Figure 2). The non-parametric modeling techniques (Appendix B) have emerged gradually into effort estimation, but applications are still in short supply.

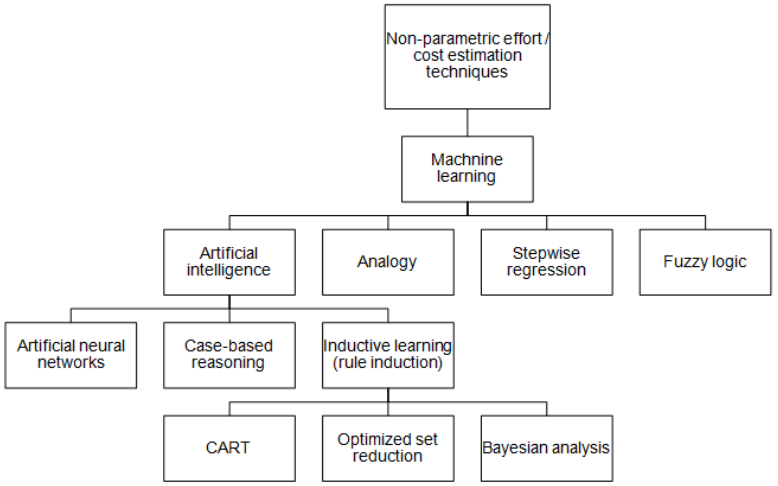

Figure 2. *The non-parametric effort estimation technique classification (adopted from (Briand, et al., 1999, 2000)*

Since the beginning of the 1990's research on effort estimation has increased and new approaches have been proposed and presented. These approaches include various machine learning

techniques and artificial intelligence. Based on the evaluations of these advanced techniques (e.g., (Finnie and Wittig, 1996; Mukhopadhyay and Kekre, 1992; Shepperd and Schofield, 1997; Srinivasan and Fisher, 1995)), one cannot conclude a superior estimation technique, although case-based reasoning seems to be a promising one out of the advanced formal approaches. The results of the evaluations depend on the case and are partially in conflict with each other. Although new approaches have been introduced, in practice the methods employed are mostly old approaches. A reason for this is that the new approaches, such as artificial neural networks, produce results hard to interpret in practice (Briand, et al., 1999).

In fact, formal techniques in general are seldom employed in the software engineering industry. Studies (e.g., (Gray, et al., 1999; Hihn and Habib-agahi, 1991; Jørgensen, 2005; Jørgensen and Sjøberg, 2004; Kitchenham, et al., 2002; Virtanen, 2003)) imply that applied techniques in the software industry are primarily informal, and based on analogy and expert analysis, particularly since there is no conclusive evidence that a formal method outperforms the informal expert analysis (Jørgensen, 2004a, 2004b, 2007; Kitchenham, et al., 2009). The analysis of fifteen studies comparing the accuracy of expert estimates with model estimates reported that findings were distributed equally, i.e., five studies were in favor of expert estimations, five studies found no difference and five studies were in favor of formal model-based estimation (Jørgensen, 2004b).

In another study (Niessink and van Vliet, 1997), the expert judgment technique has been reported to outperform the Function Point Analysis based estimations in the context of predicting maintenance effort. *Expert judgment estimation* is defined as an "estimation conducted by a person who is recognized as an expert on the task and who follows a process that is, to some degree, non-explicit and non-recoverable" (Jørgensen and Sjøberg, 2004, p. 317). Reasons for applying expert judgment instead of formal estimation methods include flexibility regarding required input for estimations and time spent on estimations.

Software project effort estimation has been increasingly studied since 1960's both in the academia and in the software industry. In particular, the research interest lies within *modeling* effort and cost estimation. As models in general, they describe a phenomenon *ex post*, and thus their use *ex ante* for estimation can be challenging; rather, they can be used for simulating the software project effort after project's completion when the actual effort is in hand. Moreover, the models require information as input parameters which cannot be obtained before the project is completed. For example, COCOMO model (Boehm, 1981) requires the number of lines of code when modeling software project effort.

Effort and cost estimation has existed in software engineering research literature since 1960's when the first models to estimate effort and cost were introduced (Conte, et al., 1986). These pioneering models included linear statistical models such as Nelson and Farr-Zagorsky. The famous formal models are from the turn of the 1970-80's, and include the Putnam SLIM model, Price-S model, the Jensen model, and the COCOMO model.

According to (Pressman, 2005; Royce, 1998; Smith, et al., 2001), the most widely known models include the COCOMO (Constructive Cost Model) (Boehm, 1981) and COCOMO II (Boehm, et al., 1995, 2000) models, which are primarily based on software size in lines of code, and the *Function Point* (FP) count (Albrecht and Gaffney, 1983). The FP count, or *Function Point Analysis* (FPA), is based on the software functionality, and can be used to derive effort with a known productivity factor, and has many variants such as Mark II FP (Symons, 1988).

The COCOMO models, being influential to our research, are described in concise in the following. Besides providing a well-defined cost estimation model, the COCOMO literature (Boehm, 1981; Boehm, et al., 2000) provides different work breakdown structures with relating effort distributions.

Boehm (1981) introduced the original COCOMO model in 1981. The original *COCOMO model* is a collection of three different models: the Basic model, the Intermediate model, and the Detailed model. The purpose of using a more complex

model is to achieve more accurate estimation by taking more factors into account. This in turn requires more details to be known. The Basic model, which can be applied in the early stage of the project, estimates the effort based primarily on the software project's size in terms of program lines of code (*kilo delivered source instructions*, KDSI). The Intermediate and Detailed models use an *Effort Adjustment Factor* (EAF) and slightly different coefficients for the effort equations. The EAF is a product of the six-scaled Effort Multiplier and the corresponding cost driver. The COCOMO cost drivers consists of fifteen independent product, computer, personnel and project attributes which determine the project's effort. The major difference between the Intermediate and the Detailed model is that Effort Multipliers are in the Detailed model used for each project phase (Agarwal, et al., 2001; Boehm, 1981; Fenton and Pfleeger, 1997).

COCOMO was enhanced to *COCOMO II* in the mid-1990's in order to develop the model to address the new needs of evolving software engineering such as distributed software and component techniques (Boehm, et al., 2000; Fenton and Pfleeger, 1997). COCOMO II categorization is not, however, suitable for all project situations, and should be adjusted via context and judgment to fit individual projects (Boehm, et al., 2000). COCOMO II model's equation for counting effort corresponds with the original COCOMO's Intermediate (or Detailed) model's equation. The amount of Effort Multipliers considered depends on the project's development phase as the knowledge on the project grows (Agarwal, et al., 2001). Moreover, besides using only lines of code for the software sizing, object or function points can be used (Fenton and Pfleeger, 1997; Pressman, 2005).

Besides estimating effort and cost, the COCOMO models provide effort distributions over different work breakdown structures. A WBS was suggested for projects employing a waterfall project lifecycle process in (Boehm, 1981). The WBS is applied with the COCOMO cost estimation model as the model estimates how effort is distributed on different activities between eight major categories, namely requirement analysis,

product design, programming, test planning, verification and validation, project office functions, configuration management and quality assurance, and manuals, each having specific activities in the four project lifecycle phases. These phases were based on earlier estimation methods and the waterfall lifecycle model (Boehm, 1981).

COCOMO II has been developed to be usable by projects employing either waterfall or spiral software engineering processes. In the waterfall approach of COCOMO II, software activity work was divided into five major categories: management, system engineering, programming, test and evaluation, and data. This breakdown was adapted from the COCOMO's eight categories, which were partly reorganized and renamed. The requirements, product design, and configuration management and quality assurance categories were organized as system engineering sub-activities. 'Verification and validation' was renamed 'test and evaluation', 'manuals' became 'data', and 'project office functions' became 'management' (Boehm, et al., 2000).

The COCOMO II spiral approach is based on the same WBS approach applied in the RUP (Royce, 1998) The RUP default WBS is based on seven main categories, namely management, environment, requirements, design, implementation, assessment, and deployment. Each of these seven categories includes activities relating to four project phases (inception and elaboration of the engineering stage, and construction and transition of the construction stage) (Royce, 1998). The COCOMO II WBS for spiral process is based much on the same seven categories, yet RUP's 'environment' became 'environment and configuration management' in COCOMO II. Also, the activities within the four phases partly differ (Boehm, et al., 2000). Moreover, COCOCO II dropped COCOMO's modern-programming-practices parameter in favor of a more general process-maturity parameter. COCOMO II was also enhanced with several new parameters (Chen, et al., 2005), for example, with the development of reuse, multi-site development, architecture and risk resolution, and team cohesion. Moreover,

the organization can add new proprietary parameters reflecting its particular situations.

2.2.2 Effort data collection

The software engineering literature (e.g., (Pressman, 2005; Royce, 1998; Sommerville, 2001)) describes effort-related functions during project execution (collection, monitoring, assessments, and re-estimations) very superficially compared to the effort-related functions during project planning (e.g., effort estimation, scheduling etc.).

However, a few descriptions exist. In the beginning of the 1980s, Boehm (1981) described a follow-up system for projects' effort (cost) information. He pointed out that in the beginning effort estimation is imperfect and actual effort data is needed for calibration and change management. Moreover, not every project fits into the estimating model. Hence, the employed formal effort estimation model requires a particular follow-up system. This system both supports effective project management and benefits the long-range effort estimation capabilities. The data collected in a consistent manner via control activities over several projects can be analyzed to determine how the actual effort distribution differs from the estimates. The differences are fed back to calibrate the model. For a new project, data collection should be considered to calibrate their estimating models (Boehm, 1981).

The importance of the effort-related functions during project execution, such as effort collection, was raised also in (Tsoi, 1999), where a framework for software project development management was proposed. This framework concentrated on the pre-project and project phases (referred as acquisition and operation phases, respectively), and especially on the monitoring the effort in a software project. The framework relied on two basic principles which occur in software projects: the existence of change due to the dynamic environment, and the need for dynamic, continuous measurement on project progress to collect real-time information.

2.2.3 Effort assessments and post-mortem analyses

The projects are assessed after a specific time period, e.g., project phase, iteration, increment, or sprint, and the project is analyzed in detail after it has been completed. This retrospective analysis is called an *assessment* or *postmortem analysis*, a *project closure* or *retrospective*, or a *post-project analysis* (Brady and DeMarco, 1994; Collier, et al., 1996; Haapio and Eerola, 2006; Jalote, 2000). The motivation of the analysis is to consider the finished project or its phase and its results carefully in order to understand and explain it. The teams and organizations want to learn from experience and collect information for future utilization, to improve software processes, and to facilitate product development based on learning. One of the goals of such assessment is to decrease the future effort estimation inaccuracies.

Effort assessments and postmortem analyses are carried out as a part of SPI. Analyzed effort information is required by the maturity models, e.g., ISO/IEC 15504 (ISO, 1993) or CMMI (SEI, 2006; SEI, 2010) as evidence of process maturity. Effort is assessed regardless of the software development model, not only in waterfall-based projects but also the spiral, incremental, iterative, and agile approaches, such as widely employed RUP (Royce, 1998) or Scrum (Schwaber, 1995; Takeuchi and Nonaka, 1986).

The effort assessment takes a retrospective into the success of effort estimation while retrospective analyses are performed to get exact data from actual effort and project results in terms of requirements fulfillment of the stakeholders. A requirement is defined as "something that the product must do or a quality that the product must have" (Robertson and Robertson, 1999, p. 5). The software product holds both functional and non-functional requirements (Robertson and Robertson, 1999), and on the other hand, both user and system requirements (Pressman, 2005). The functionality of the product, i.e., things that the product must do, and the data manipulated by the functions, is stated as the functional requirements. The non-functional requirements, on the other hand, are the product's qualities, i.e., the requirements

for look and feel, usability, performance, operational, maintainability and portability, security, cultural and political, and legal (Robertson and Robertson, 1999). The user requirements are statements of "what services the system is expected to provide and the constraints under which it must operate" whereas the "system requirements set out the system services and constraints in detail" (Sommerville, 2001, p. 118).

The rationale for performing assessments and postmortem analyses is well-represented in the literature as an essential part of software engineering process and organization's knowledge management (Rus and Lindvall, 2002), and highly endorsed by both IT consultants (Iacovou and Dexter, 2005) and scholars. To put it simply, the analysis can increase the odds of project success (Verner and Evanco, 2005) and improve project cost estimation (Birk, et al., 2002). Despite the benefits, the minority of the organizations employ assessments consistently (29% according to (Verner and Evanco, 2005)). The proposed analysis processes may require rather large resources (both time and personnel) which, in practice, are quite limited in the software industry. Moreover, in many cases the analysis descriptions provide only general guidelines without detailed methods. As the general software engineering literature (e.g., (Sommerville, 2001)) attempts to cover the whole spectrum of software engineering, it provides understandably only rough descriptions of assessments and postmortem analyses. More detailed presentations can be found in research papers.

In 1990s, the interest in postmortem analysis increased in particular (Brady and DeMarco, 1994), moving from analyzing the developed system (Kumar, 1990) to analyzing the process how the systems were developed (Collier, et al., 1996), and from closing the project to improve the software engineering process. These software project postmortem analysis descriptions were complemented with e.g.:

- a proposal to separate postmortem analyses for planning, design/verification, and field use (Tiedeman, 1990)

- an online presentation of an approach to conduct postmortem analysis (Kerth, 1998)
- a presentation that emphasize learning from documented success stories (Nolan, 1999)
- a proposal for a light-weighted postmortem analysis method for small and medium size companies to increase analysis consistency (Dingsøyr, et al., 2001)
- a book much referred to on project postmortem analyses (Kerth, 2001)
- a distinction between two postmortem analysis types (Birk, et al., 2002): one is a general postmortem analysis that collects available experience from an activity and the other is focused in understanding and improving a project's specific activity, such as effort estimation
- a comparison of two types of postmortem outcomes: reports and stories (Desouza, et al., 2005a, 2005b)
- different postmortem approaches (Dingsøyr, 2005)
- a postmortem method for the telecom domain (Sertic, et al., 2007)
- retrospective applications for agile software development (Kinoshita, 2008; Maham, 2008)
- postmortem analysis for maintenance processes (Anquetil, et al., 2007).

Conducting postmortem analyses have also been criticized. Postmortem analyses, by nature, take a reactive approach, i.e., analyze a project after it failed, while proactive actions would be more advantageous in preventing the project from failing (Awazu, et al., 2004). Assessing effort during the project is one of these steps; it enables necessary, even drastic, measures be taken before the project is completed. It is also reminded that documented and recorded information does not improve anything unless the knowledge is shared and available, and the tools attract to utilize the information (Lyytinen and Robey, 1999; Petter, et al., 2007).

3 Research methodology

In this chapter, we discuss our research process and objectives, and present the key research problems with the employed methodologies to address these research problems.

3.1 RESEARCH PROCESS, PROBLEMS AND OBJECTIVES

Our research process is based on the principles of the constructive research methodology. The constructive research method is described in sub-chapter 3.2.1. Besides constructive research method, we employ several other research methods to support our research process; the case study and action research methods to be most important. These supportive research methods are described in sub-chapter 3.2.2.

With constructed research artifacts, we attempt to solve organizational problems related to effort and its estimation (Figure 3). This thesis considers the *unit of analysis*, effort, in the context of software development projects.

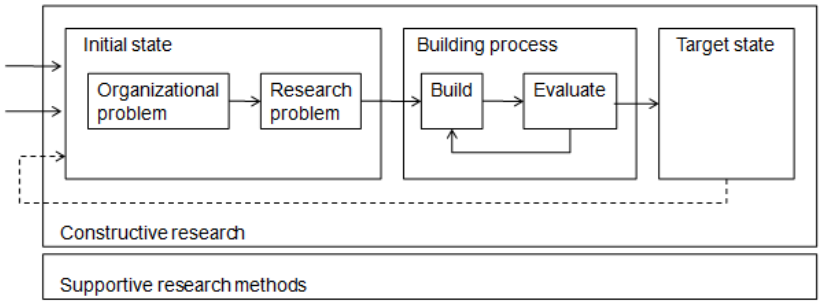

Figure 3. *The research process (adopted from (Järvinen, 2001))*

The motivation for building an artifact is either lacking of that artifact or low quality of the outcomes achieved with the

old artifact (Järvinen, 2001). The research artifacts are built in the building process. The purpose of the building process is to achieve a transition from the initial state to the target state (Järvinen, 2001). In initial state, we transform the organizational problems into research problems. In the iterative building process, a research artifact is first build and then evaluated. If the evaluation criteria are met, the research artifact can be considered reach its target state. On the other hand, if there is still room for improvement, we return to the building activity where the artifact is upgraded. In time, the target state does not satisfy the needs anymore, and the state is returned to initial.

In addition to the problems found in organizations and by the software engineering research that were described in the Theoretical background chapter, we include in our research organizational problems that are observed at the research site. It is notable, that the research process we apply is iterative, supplementary and cumulative, i.e., new organizational problems are observed at the research site and identified as a result from an on-going research iteration, and new iterations are re-initiated for each new problem that we define as a research problem. In the following, we formulate the research problems we aim to solve in this thesis.

Several effort and cost estimation models and methods have been proposed during the last decades. Despite the numerous applications available for estimation, effort is, however, frequently underestimated (Gruschke and Jørgensen, 2008; Kitchenham, et al., 2009). As mentioned in the Introduction chapter, according to studies (e.g., (Briand, et al., 1999; Shepperd, et al., 1996)), the effort estimation methods and tools have reached only low accuracy levels with MRE-values worse than 0.30, whereas the target value between the estimated and actual effort for software industry companies can be three times smaller, for instance.

The effort estimation research and the different estimation techniques emphasize two factors in respect to the difficulty in producing accurate estimates: software size and the available data for estimations (Armour, 2002). Most formal effort and cost

estimation models and methods require a determination on the size of the software to be produced. The size is generally determined for example in terms of *lines of code* (LOC), *function points* (FP), or *use case points* (UCP) (see sub-chapter 2.1 Key concepts). The prediction of the size of the final software product is, however, challenging and complex. Another factor which reflects to inaccurate effort estimates is the shortage of knowledge on the software to be produced since these estimates are made especially in the early stages of the software development process (Armour, 2002; Boehm, 1981).

Concluding from the effort-related challenges described above and in the Theoretical background chapter, we can formulate our first and primary research problem as follows:

"Effort (and cost) estimates are inaccurate." (RP1)

Although several formal approaches have been proposed for effort estimation they are seldom employed in practice. As mentioned in the sub-chapter 2.2.1 of Theoretical background, the applied effort estimation approaches are in most cases informal, and based on analogy and expert analysis. Jørgensen daunts that it might not be possible to develop effort estimation models that replace expert judgment, and therefore the best effort estimation improvement strategy may be to improve the judgment-based effort estimates (Jørgensen, 2005).

As assumed in the Introduction chapter, an overlooked, possible factor explaining poor effort estimates is the light consideration on different activities involved with the project. Therefore, the likelihood of unintentionally leaving out significant activities affecting the estimation is significant. Also, the focus in effort estimation research and estimation methods has been on software construction and project management (MacDonell and Shepperd, 2003), whilst presenting the other activities related to the project merely as a fixed proportions from the software construction effort, and as effort overhead. These activities, which are not directly related to software construction or project management, include various

management and support activities, each carried out by several members of the project.

Experiences from the software industry imply that in effort estimations where expert judgment technique has been applied the estimators consider exclusively those activities that occur for them in a software project and focus on the effort of actual software construction. Indeed, research (e.g., (Jørgensen, 2004b, 2005; Jørgensen and Sjøberg, 2004)) confirms this observation. Also, there is a correlation between years of experience and the variety of different project activities considered for the estimate (Jørgensen and Sjøberg, 2004): more experienced experts consider a larger variety of project activities. Different experts estimate different areas, e.g., programmers estimate the amount of programming effort.

Hence, we can formulate our second research problem as follows:

"Estimators consider mostly core construction activities, relevant in their work, when estimating." (RP2)

A reason for the effort estimate inaccuracy can be the inadequacy of previous projects' effort data collection when effort is both reported badly and collected without analyzing it properly afterward. The collected effort data is used for both improving team performance in upcoming project phases and project and in calibrating the weights that adjust the factors and drivers which are used for the effort estimate derivation in the organization in question.

The proposed processes for post-mortem analysis (e.g., (Collier, et al., 1996)) describe post-mortem of the whole project, and provide general guidelines without a detailed method to analyze effort. Furthermore, as noted in sub-chapter 2.2.3, the proposed analysis processes require rather large resources (both time and personnel) which, in practice, are quite limited in the software industry.

Hence, we can formulate our third and fourth research problems as follows:

"Effort data quality is bad, i.e., it is not reliable." (RP3)

"Effort data is seldom assessed." (RP4)

An important topic to consider in improving effort estimation is the overall management of software project effort since effort estimation is not an isolated activity. As mentioned in the sub-chapter 2.2.1 of the Theoretical background, effort has not been seen as an independent area of management like risk or quality. The software engineering literature (e.g., (Pressman, 2005; Royce, 1998; Sommerville, 2001)) discusses risk management, quality management and configuration management individually but effort is covered as parts of software project management. The literature describes the effort-related functions during project execution (collection, monitoring, and re-estimations) very superficially compared to the effort-related functions during project planning (e.g., effort estimation, scheduling etc.).

Hence, we can formulate our fifth research problem as follows:

"Effort-related functions are presented in a scattered way." (RP5)

It has been argued that the formal COCOMO models and Function Point count are too complex and uncertain for practical use (Kemerer and Porter, 1992; Rask, 1992; Sommerville, 2001; Symons, 1988). The applications of these effort estimation models are usually not transparent, i.e., the factor weights used for effort derivation are not obtainable in order to validate them. Transparency is required by the estimator to evaluate the reasonability of the gained estimate, i.e., what comprises the effort and the costs.

Hence, we can formulate our sixth and final research problem as follows:

"The effort estimation models, methods and applications are 'black boxes', i.e., the estimator cannot judge the result." (RP6)

For addressing these six research problems (Table 1), we employ constructive research methodology as our primary methodology.

Table 1. *The research problems and the related aims*

Research problem (Initial state)	**Aim (Target state)**
RP1: *"Effort (and cost) estimates are inaccurate."*	Increase in the likelihood for more accurate effort (and thus cost) estimates has been enabled.
RP2: *"Estimators consider mostly core construction activities, relevant in their work, when estimating."*	Increase in the estimator consideration of all significant project activities has been enabled.
RP3: *"Effort data quality is bad, i.e., it is not reliable."*	Increase in the reliability (and thus quality) of the effort data has been enabled.
RP4: *"Effort data is seldom assessed."*	Increase in the ratio and quality for effort data assessment has been enabled.
RP5: *"Effort-related functions are presented in a scattered way."*	Increase in the opportunities to obtain comprehension on effort-related functions.
RP6: *"The effort estimation models, methods and applications are 'black boxes', i.e., the estimator cannot judge the result."*	Increase in the opportunities for the estimator to question the estimate.

The objective of this research is to develop and describe an effort management process and suitable artifacts, e.g., models and methods, which can be used to improve the estimation accuracy and which address the other research problems. Accurate estimates can increase customer satisfaction, customer and supplier business profitability, and the well-being of the personnel. The results should be advantageous to be applied both with expert judgment estimations and estimates made with formal methods.

3.2 RESEARCH METHODS

Several different research strategies have been proposed for the software engineering and information systems research. These research methodologies can be found very useful, but no-one is sufficient by itself to form a well-grounded research program. In fact, Nunamaker, Chen and Purdin (1991, pp. 95-96) state: "where multiple methodologies are applicable, they appear to be complementary, providing valuable feedback to one another."

Nunamaker, Chen and Purdin proposed a multimethodological approach to information systems research, where the four research strategies have an effect to each other (Nunamaker and Chen, 1990; Nunamaker, et al., 1991). These four research strategies were theory building, experimentation, observation, and systems development. Our research applies all of these research strategies. However, we do not employ the multimethodological approach as defined in (Nunamaker and Chen, 1990; Nunamaker, et al., 1991). Rather, we apply a methodology where it is most suitable, and mostly together with the case study methodology. We apply case studies as the observation methodology to both gather information about research problems and to evaluate research results in practice. The theory building strategy is applied with all research problems. The research products are evaluated with prototype experiments in the software industry. Prototyping is used as a proof-of-concept to demonstrate feasibility (Nunamaker, et al., 1991).

Our research is primarily based on constructive research, or systems development (Nunamaker and Chen, 1990; Nunamaker, et al., 1991). In the system development research process that we apply, we first construct a conceptual framework related to the software process improvement of effort management. The relationships among the elements of effort management are then defined, and after analyzing and designing a research artifact a prototype is build. The prototype is employed in case studies to observe and evaluate the artifact.

The system development research process is both parallel and iterative.

Our research is explorative. In explorative research, different research strategies can be for example an exploratory survey, experiment, or case study (Yin, 1994). Moreover, the research includes characteristics of *action research* (sub-chapter 3.2.2): collaboration, implication and situation between the research and the project under examination.

The research is carried out as a part of organization's software process improvement activity. Such action research can be considered as a part of *constructive research* (sub-chapter 3.2.1) where both building and evaluation closely belong to the same process (Järvinen, 2001). In software industry, an industry-based research realizes often as a SPI activity. In *industry-based research*, the research is conducted within software industry by a researcher who is affiliated with the industry. It is noted that "new knowledge is increasingly produced through a variety of work- and industry-based research practices" (Garrick, et al., 2004, p. 329).

Research based on industry data does not come without challenges. In (Haapio and Menzies, 2009), we describe challenges we faced during the data mining experiment in the study of Paper II, and how we overcame them. These challenges of ours originated from the data that was limited inside, noisy, and skewed with local factors.

3.2.1 Constructive research

To address the research problems, we employ constructive research methodology (Iivari, 1991) as our primary methodology. Constructive research is a theory building research methodology. The theory building research strategy includes development of new ideas and concepts, and construction of conceptual frameworks, new methods, or models (Nunamaker and Chen, 1990; Nunamaker, et al., 1991).

Constructive research (Järvinen, 2001), also referred to as *design science* (Hevner, et al., 2004; March and Smith, 1995) or *systems development* (Nunamaker and Chen, 1990; Nunamaker, et al.,

1991), consists of two basic activities (Figure 3): building products and evaluating them (Järvinen, 2001). The evaluation of the built product is based on user value or utility, i.e., feasibility is demonstrated when the created research artifact (product) serves human and organizational purposes (Järvinen, 2001; March and Smith, 1995).

There are four research products; constructs, models, methods, and instantiations (Järvinen, 2001; March and Smith, 1995). *Constructs* (or concepts) form the domain vocabulary. In this research, the different elements of the effort management refer to constructs. A *model* expresses the relationships among these constructs. Here, the effort management frameworks refer to the model. A *method* is a defined set of steps used to perform a task; here, the proposed stepwise effort assessment method. An *instantiation* is the artifact realization in its environment. In this research, the instantiation is created with case studies in which the proposed effort management framework and its related improvement areas are employed to custom software development projects.

Evaluation is the key activity for assessing constructive research (Järvinen, 2001; March and Smith, 1995). Nunamaker, Chen and Purdin (1991, p. 95) warn that because "of the emphasis of the generality, the outcomes of theory building often display limited practical relevance to the target domain." The evaluation process attempts to diminish this concern by applying suitable metrics and comparing the performance of artifacts for specific tasks (Järvinen, 2001; March and Smith, 1995). In other words, the constructive research results need to be pragmatic and advantageous to be utilized in the industry, and they must provide significant improvement (Järvinen, 2001; March and Smith, 1995). The various evaluation criteria of novel research artifacts stated by March and Smith (1995) and complemented by Järvinen (2001) are applied, when possible, in this research (Table 2).

Table 2. *The research evaluation criteria (adopted from (Järvinen, 2001; March and Smith, 1995))*

Research outcome	Metrics
Construct	Completeness, simplicity, elegance, understandability, and ease of use. Communication and cognition.
Model	Their fidelity with real world phenomena, completeness, level of detail, robustness, and internal consistency. Form and content, and richness of knowledge representation.
Method	Operationality, efficiency, generality, and ease of use. Application domain.
Instantiations	The efficiency and effectiveness of the artifact and its impacts on the environment and its users. Emergent changes with positive and negative unanticipated outcomes, in addition to economic, technical and physical impacts on social political and historical contexts.

As applied sciences, computer science and software engineering are especially suitable for design science and the constructive research methodology (Iivari, 1991; Järvinen, 2001; March and Smith, 1995). The research intent is a reason for differentiation between design science and both natural and social science (Järvinen, 2001; March and Smith, 1995). The intention of our research is to change the current state of the accuracy of effort estimates. Järvinen (2001) generalizes the differentiation between natural and design sciences to 'is' and 'ought to', respectively. To change an unwanted state in software industry to a desired state, design science provides a well-suited framework for the transition.

In fact, we have applied both Järvinen's (2001) and March and Smith's (1995) taxonomies of research methods in determining our primary research methodology. While we are studying reality, we do not focus on the phenomena behind the reality but instead seek improvements and utility of the research results. In Järvinen's taxonomy, the improvements and utility can be achieved with the approaches for building and evaluating innovations. Constructive research combines these two approaches under one research methodology. In addition to

Järvinen's taxonomy for differentiation between sciences, March and Smith's research framework identifies different types of design science products (constructs, models, methods and instantiations) which we use in our research (Järvinen, 2001; March and Smith, 1995).

3.2.2 Supportive research methods

Case study

The primary supportive research method for this study is case study. Case studies are employed to both examine the data and to evaluate the research artifacts.

Case study is a form of the observation research strategy (Nunamaker and Chen, 1990; Nunamaker, et al., 1991). Observation is often used when relatively little is known from the research domain. Nunamaker, Chen and Purdin note that since "research settings are more natural, more holistic insights may be gained and research results are more relevant to the domain under study." (Nunamaker, et al., 1991, p. 95). Case studies can be used for gathering information about a phenomenon which results as a research problem or question. In this study, the case study methodology is understood and applied in a broader context than that Yin's (1994). All research problems are more of exploratory nature rather than explanatory. We chose exploratory approach over explanatory approach from pragmatic reasons to achieve advantageous results in the software industry, i.e., we attempt to solve problems rather than to understand the phenomena behind the problem.

As the research interest lies in the comprehension of the meaning of action, case studies are used to interpret (Järvinen, 2001) effort in software projects. Moreover, we apply case studies to research evaluation in which they can be used for (Fenton and Pfleeger, 1997; Kitchenham, et al., 1995). For example, the stepwise effort assessment method is constructed by applying the constructive research methodology. A case study is employed to evaluate the method in the software

industry. This same dualistic research methodology approach is employed in constructing the two frameworks used for both improving effort management and actual managing the effort in software projects.

The empirical findings of the research analyses are based on samples of real custom software development project data supplied by a software company. The case studies are conducted at Tieto Finland Oy, a part of Tieto Corporation, which is the largest *Information Technology* (IT) services company in the Nordic countries. Tieto Corporation has approximately 18.000 employees and activity in 30 countries worldwide. Tieto is building a common business system utilizing reference model CMMI. Earlier, due to company diversity and acquisitions, Tieto applied both CMM and SPICE (ISO/IEC 15504). There are several internal research development projects on-going as a part of software process improvement including studies improving effort management in the software engineering process.

Research data consists of 32 custom software development and enhancement projects which took place in 1999-2006. These projects were delivered to five different Nordic customers who operate mainly in the telecommunication business domain. The delivered software systems were based on different technical solutions. However, the two most common technologies were based either on J2EE or on transaction management based client/server technology. The duration of the projects was between 1.9 and 52.8 months. The projects, which were carried out by different teams within the same Finnish business division, required effort between 276.5 and 51,426.6 person hours. The staffing of the teams changed from project to project.

All the projects used a waterfall-like software development process, since iterative and agile processes have emerged in more recent projects. The project process lifecycle started from either the analysis phase or the design phase. All projects ended in customer acceptance of delivery. This span forms the project's lifecycle frame, which has been quite typical for delivery software projects in this organization. The normal work iteration

is included in the effort data. Effort caused by change requests, however, is excluded from the data, because the work caused by change requests is not initially known and is therefore not included in the original effort estimation or project's scope.

The information related to the effort estimates is gathered for this research from tender proposal documents, contract documents, and final report documents. The actual effort data is gathered from the organization's previous time-booking system. The project team members enter their effort in this system.

Action research

Another supportive research methodology which has influenced our research, and which has a close relation to practice, is action research (Avison, 2002; Avison, et al., 1999; Baskerville, 1999; Baskerville and Wood-Harper, 1996; Greenwood and Levin, 1998; Järvinen, 2007; Kemmis and McTaggart, 2005; Lee, et al., 2000; Stringer, 1999). *Action research* "is social research carried out by a team encompassing a professional action researcher and members of an organization or community seeking to improve their situation" (Greenwood and Levin, 1998, p. 4). Järvinen (2007) claims, that action research methodology is similar to design science (i.e., constructive research). In his fit analysis, Järvinen compared different pairs of action research characteristics to ones in design science and concluded that the similarities are obvious and thus they should be considered as similar research approaches. Indeed, several fundamentals and characteristics of action research can be found in this thesis, too. However, a major requirement does not hold for our research: our research is industry-based, i.e., the author is affiliated primarily with the industry and only secondary with the academia. Usually, action research is conducted other way around. Therefore, our research cannot be considered as action research *per se* but as a *software process improvement* activity.

Other employed supportive research methods

Besides case study and action research, other supportive research methods were employed for the research papers of this

thesis. In addition to case study, other qualitative research method techniques (e.g., elements of the grounded theory (Glaser and Strauss, 1967)) were used in analyzing the texts of the research material (project final reports, project plans, and tender proposals) in order to determine our other unit of analysis besides effort: the different general software project activities (referred as non-construction activities). The grounded theory method was also used to form the work breakdown structure of the non-construction activities. Besides grounded theory, an *exploratory survey* was applied as a qualitative research methodology. *Structured questionnaires* were employed as the data gathering technique in the survey. Questionnaire is one of the mostly used data gathering techniques in the survey studies, a form of field methods. *Survey research* is not about administering questionnaires, but a methodology which involves gathering information for scientific purposes from a sample of population using standardized instruments or protocols (Järvinen, 2001); here, using questionnaires as data collection technique. The first questionnaire used open-ended questions (theory-creating research approach) and the latter questionnaire close-ended questions (theory-testing research approach) (Järvinen, 2001). Moreover, the study contained elements of *interpretive field study* (Klein and Myers, 1999) since our aim is to understand both the context of effort registration, and the process influencing it.

Quantitative techniques were used in a *controlled experiment* (Juristo and Moreno, 2001; Järvinen, 2001; Wohlin, et al., 2000) for exploring and analyzing the actual effort and the relation between estimated and actual effort. In a controlled experiment, as many factors as possible belonging to the studied phenomenon are under researcher's control (Järvinen, 2001). Suppositions, assumptions, speculations and beliefs are tested with experimenting. In other words, the purpose of experimentation is to match ideas with reality (Juristo and Moreno, 2001).

Experimentations have two levels (Juristo and Moreno, 2001): the first level is experimenting in laboratory (controlled

conditions), and the second level is experimenting in reality (uncontrolled conditions), i.e., experimentation is carried out with real projects. The tightness of control is, however, usually in opposition to the richness of reality at the same level of knowledge (Järvinen, 2001). In fact, the researchers have to ultimately make a trade-off between these two iso-epistemic attributes.

In a controlled experiment, the Goal/Question/Metric paradigm provides a useful framework (Wohlin, et al., 2000). In the *Goal/Question/Metric* (GQM) paradigm (Basili and Rombach, 1988), data collection is designed thus (van Solingen and Berghout, 1999):

1. Conceptual level (*Goal*): a goal is defined for an object for a variety of reasons, with respect to various models of quality, from various points of view and relative to a particular environment.

2. Operational level (*Question*): a set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.

3. Quantitative level (*Metric*): a set of metrics, based on the models, is associated with every question in order to answer it in a measurable way.

One of the techniques used to achieve goals in a controlled experiment is data mining. Data mining is also one of the popular processes for producing *business intelligence* (BI) information, which is utilized in improving the software process quality of cost or effort estimations, for instance. *Data mining* is used to reveal hidden patterns in unstructured data to provide valuable information for business utilization. Indeed, BI and data mining have been increasingly employed in the software industry for finding predictors from data for modeling software project effort. The data mining tools are employed to model the data (Pyle, 1999) by regression or with trees, for example. The models of data behavior are generated with data mining learners, using appropriate data. The data can be gathered either manually or automatically. On a general level, project data

collection can be divided into two groups according to its purpose: the data which is collected during a project for project's own purposes, and the data which is collected from multiple projects for BI and SPI purposes. Whereas the automated data gathering processes can produce a vast amount of data in some business areas (Pyle, 1999), the manual data gathering results usually in smaller and noisier data sets. Another reason for the differences in data set sizes and quality is that in non-commercial organizations government funding can enable a more research-motivated and extensive data collection, whereas in the software industry data collection is more driven by the customer needs and the process maturity models which the companies are committed to (Haapio and Menzies, 2009).

4 Improving effort management

In this chapter, we present the summary of our research papers on improving effort management in software projects. We begin with what relation the research papers have in respect to each other, and continue with the summaries of the five research papers. Then, we evaluate our results in respect to the research problems, alternative solutions, research evaluation criteria for constructive research artifacts, and the deployment of the research results at the research site. Finally, we discuss what contributions this thesis has to offer.

4.1 RELATION OF RESEARCH PAPERS

In this sub-chapter, an overview on the research publications is given. Moreover, we conclude in describing how the research papers answer to the research questions.

This thesis consists of five research papers. The first paper introduces a research framework for improving the software project effort management in software engineering industry (Paper I). The three next papers describe a series of actions to improve effort management, and thus influence the effort estimation accuracy: improvements for effort estimation (Paper II), improvements for effort data quality through improved data collection and time-booking (Paper III), and improvements for effort assessment (Paper IV). Moreover, the results of these papers relate to each other. The relationships between the papers are presented in Figure 4. Each research artifact creates output that serves as input to the next artifact in the cycle. The effort-related functions presented in papers II-IV are all parts of

the effort management framework, which we introduce in our last paper (V).

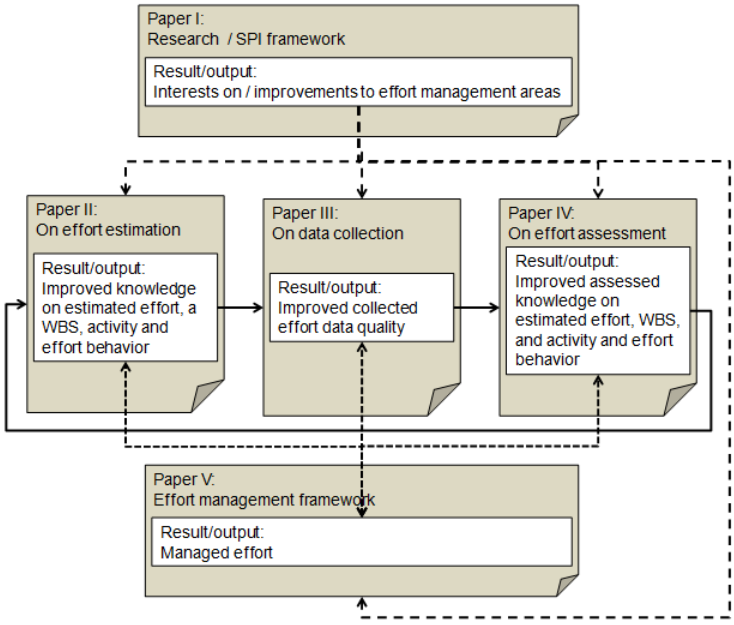

Figure 4. *The relation of research papers*

The research framework, presented in Paper I, and the research problems guided us in studying four concepts related to effort management by raising possible improvement interests in different effort management areas. The results of these studies are presented in individual papers (II-V).

Paper II results in improved knowledge on software project activities and their effort. These results including new activities and set of activities require efficient adoption by the project team and thus are input for the consideration of data collection practices (Paper III) to improve effort data quality. The collected effort data is used in effort assessment (Paper IV). The assessed effort, based on improved effort data quality achieved with Paper III's results, closes the cycle and is input for effort estimation method's calibration and project team's software project activity and effort knowledge, which is thus increased.

4.2 SUMMARIES OF PAPERS

In this sub-chapter, we briefly present the main ideas and research results of our studies.

4.2.1 Improving effort management research process

The aim of our research is to improve effort management to increase effort estimation accuracy. In Paper I, we propose a research framework for improving software project effort management as a part of an organization's software process improvement activities. We consider effort management to include functions which are necessary to manage the effort consumed by a software project. These functions include effort estimation, effort collection, and effort analysis[1].

In Paper I we strive to present the software project lifecycle and its effort-related and activity-related functions as a coherent effort management process, and provide a research framework for improving that process. Two perspectives for managing effort are presented. The two perspectives include effort distribution on particular activity sets and the necessary functions during different software project phases. Moreover, the general software project activities, later referred as non-construction activities, are defined.

The first perspective in improving effort management is to consider necessary functions during different project phases. The different phases include a phase prior to a software project, the three phases of the software project lifecycle (pre-project, project, and post-project), and a phase after the project does not exist anymore. The effort-related and software project activity related functions to be considered in SPI work are listed in Table 3.

[1] In Paper IV, we complemented effort analysis function with effort assessment.

Table 3. *The effort-related and activity-related functions in effort management*

Project-related phase	**Functions related to software project's**	
	Effort	**Activities**
Prior to project's existence	Estimation method preparation	Organization's default WBS establishment
Project: Pre-project	Estimation	Project's WBS creation in effort time-booking system
Project: Project	Collection, Monitor, Assessment[2], Re-estimation	Project's WBS adoption, Effort registration
Project: Post-project	Analysis	Analysis
After project's existence	Estimation method adjustment	Organization's default WBS adjustment

The other perspective, the effort distribution, is for dividing the whole software project effort into reasonable activity sets, which assists the concentration on the activity sets needing the most attention for improvement. Based on our previous research (Haapio, 2004) we propose that the effort of software construction, project management, and non-construction activities be analyzed separately to identify characteristics typical for each category, as it aims to minimize the fluctuation within these categories between projects. The rationale and the original breakdown into these three categories presented in (Haapio, 2004) were based on the flexibility requirements of adding any new future project activity into one of the three categories according to its function and the actor performing the activity. The function of a software project activity can relate to software construction, to management, or to other. A software project consists of two distinctive types of actors, a project manager and project team members. Both actors can construct software whereas the project manager solely leads a project.

[2] In Paper IV, we complemented effort analysis function with effort assessment.

However, various management activities can be conducted also by project team members. Hence, a software project includes three major activity categories:

- *Software construction* involves the effort needed for the actual software construction in the project's lifecycle frame, such as analysis, design, implementation, and testing. Without this effort, the software cannot be constructed, verified and validated for the customer hand-over.
- *Project management* involves project lead activities that are conducted solely by the project manager, such as project planning and monitoring, administrative tasks, and steering group meetings.
- All the general software project activities in the project's lifecycle frame that do not belong to the other two main categories can be termed *non-construction activities*. These activities include various management and support activities such as configuration management, customer-related activities, documentation, orientation, project-related activities, quality management, and miscellaneous project activities, which are carried out by several members of the project.

Hypothetically, many of the non-construction activities can be eliminated from a software project and the software can still be constructed. In practice, however, a project would be more or less uncontrolled and unsupported without the non-construction activities. This in turn can increase both the software construction effort and project management effort needed to get the project accomplished.

In our effort management related SPI work, we have been applying the two perspectives of effort distribution on particular activity sets and necessary functions during different phases together in a matrix (Figure 5), concentrating especially on the non-construction activities. The matrix assists to both improve the management of the whole area of effort (all elements) and,

on the other hand, to focus on sub-areas which need a special improvement interest on a particular moment (one element).

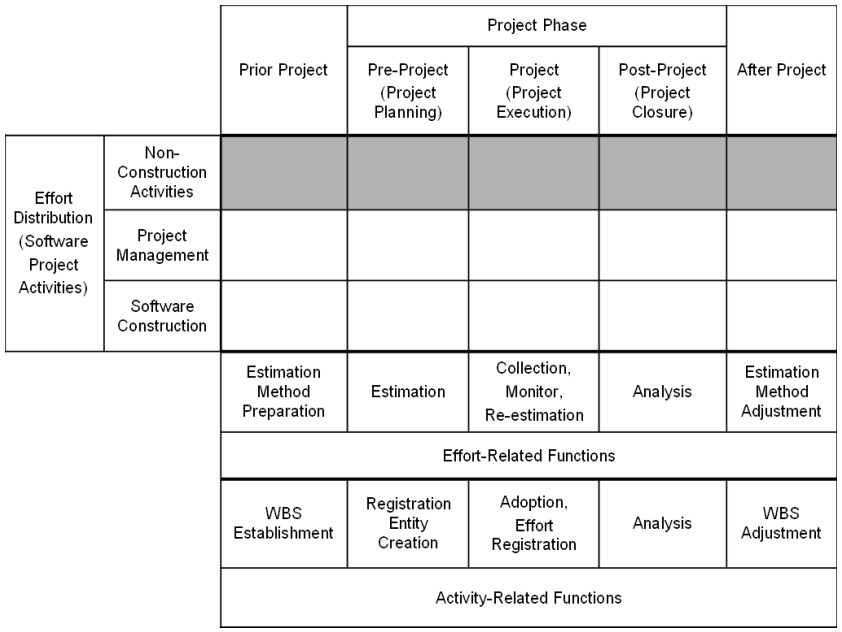

Figure 5. *Matrix for improving effort management in software projects*

Our approach is dynamic considering the effort-related functions, and the employment of the approach depends on the needs of the organization which wants to improve its effort management instead of concentrating merely on the effort estimation function.

4.2.2 Improving effort estimation

In order to understand the impact of the non-construction activities have on effort estimation and the estimate accuracy, we examined the effort of the three main software project activity categories (software construction, project management, and non-construction activities) in Paper II. The category in particular focus was the non-construction activities of which activities were individually analyzed.

In Appendix A, the key figures of the effort proportions on analyzed 32 software projects' activity sets and the non-construction activities are presented. The median effort of

software construction does not differ from that reported by MacDonell and Shepperd (2003) (76.7% and 76.9%, respectively), although the approach and basis of division were different to ours. This suggests that there might be a tendency to have a certain effort distribution, and that the division in this study seems to be appropriate. Also, it is notable that in this data set the non-construction activity effort proportion is very similar to that of the project management (Figure 6). This suggests that it is important to view the non-construction activities and their effort as an independent group.

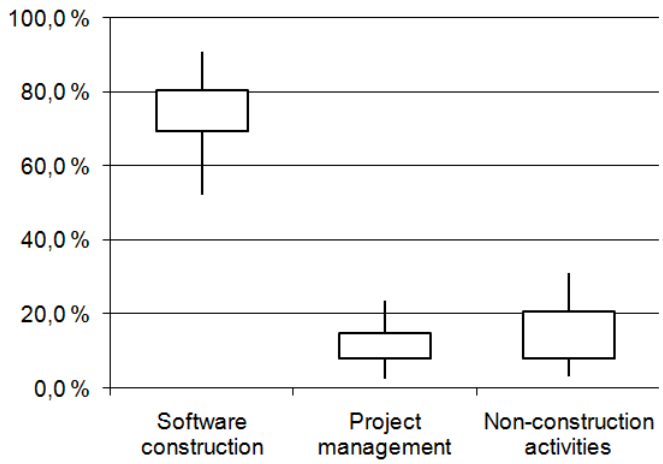

Figure 6. *The effort of software construction, project management and non-construction activities in the analyzed 32 software projects*

Not only the effort of the non-construction activities comprises a significant amount of effort (median 11.2% is comparable with project management's 11.3%) but also the effort of the non-construction activities results in a great variance between projects if the activities are not planned and managed (i.e., controlled) during the project. For example, the 32 custom software development projects considered in Paper II comprise a deviation of 7.68% (Appendix A). This deviation causes deviations to other main activity categories' effort.

In previous research (Haapio, 2004, 2006), by applying the grounded theory methodology, we have identified the different

non-construction activities in the case studies concerning our data set of 32 custom software development projects from the research site. The activities are generic and most exist in every custom software development project. We have defined the following current non-construction activities in Paper II:

- Configuration management
- Customer-related activities
- Documentation
- Orientation
- Project-related activities
- Quality management
- Miscellaneous activities.

In addition to these activities found in our data set, other software project activities can be considered as non-construction activities based on the definition of the activity category (sub-chapter 4.2.1). These activities relate to global sourcing projects (e.g., project start-up or culturally-related tasks), to open source projects (e.g., license management), or to product line projects (e.g., product line management), for instance.

In Paper II, we seek for predictors of software project activity effort related to the three main software project categories and the six non-construction activities by using the data mining technique. We exclude the 'miscellaneous activities' response class for two reasons: firstly, the frequency of these activities appearing in a project was small (28.1%) compared with the frequencies of the other activities (53.1%-90.6%), and secondly, 'miscellaneous activities', with no common denominator, is a 'dump' category. We applied several machine learners provided by the Weka application (Holmes, et al., 1994) to mine the predictors. First, we applied learners for continuous classes. However, the results for continuous classes turned out to be somewhat disappointing, with small correlation coefficients. We continued by discretizing the classes, and applied the learners for discrete classes. Finally, all learners for discrete classes achieved a result when applying *Feature Subset Selection* (FSS) prior learning.

Our data mining experiment results are two-fold: whereas we find no evidence that none of the three main software project activity categories is a significant factor influencing effort estimation, we can find evidence of a relation between the estimated total software project effort and one of the non-construction activities, namely quality management, actual effort. This finding can be used in improving effort estimations, and is a useful supplement to the scanty research on the possible impacts of effort estimates on project work and behavior (Jørgensen and Sjøberg, 2001).

The practical implication of this study relate to the specific conclusion we can offer to software businesses and quality management. The implication is that the actual *quality management* (QM) effort proportion of total project effort will be larger in projects that are estimated to be small rather than large, which has to be considered in the effort estimates.

Quality management is essential to software projects and their success. Intuitively, absolute QM effort increases with project size and total effort, as there are more project deliverables requiring quality assurance. Indeed, our data shows such a tendency between the absolute effort values of quality management and total project effort. However, our data does not support a pattern of a linear, logarithmic or exponential growth of quality management effort as the total effort of a project increases. This implies that quality management effort is not a constant in software projects and thus cannot be predicted with an approach applying merely a constant. In fact, this supports the findings for quality assurance effort behavior in software projects (cf. Figure V.24 in (Abdel-Hamid, 1984, p. 419)).

We can conclude that we can set a maximum proportion for QM effort as a rule-of-thumb when estimating the QM effort for large projects. In a tight market situation, the correct smallest possible estimate of the QM cost could improve the chances of winning a deal. However, the rule-of-thumb does not hold for smaller projects. It seems that in small projects we can set a

minimum proportion for the QM effort but the actual amount can turn out larger.

Our finding suggests that projects that are perceived to be smaller at their beginning seem to devote more effort on QM activities than larger projects (Figure 7). This, in turn, can result in better quality projects, also in terms of actual effort (i.e., cost) estimates, promoting project success (cf. (Standish, 1999). In other words, a small project might not consume all a team's energy in the software construction tasks. Instead, the team can devote energy also to quality management. Vice versa, the results imply that projects that are perceived to be larger at their beginning tend to spend less effort on quality management as a proportion of the total project effort. One reason for this might be that the team's energy is consumed by other (i.e., construction) project activities. In fact, there is evidence that effort estimates have an impact on software project work and behavior whether a project is estimated, and thus perceived, as either a small or large project when the project is starting (Abdel-Hamid, 1984, 1986; Jørgensen and Sjøberg, 2001).

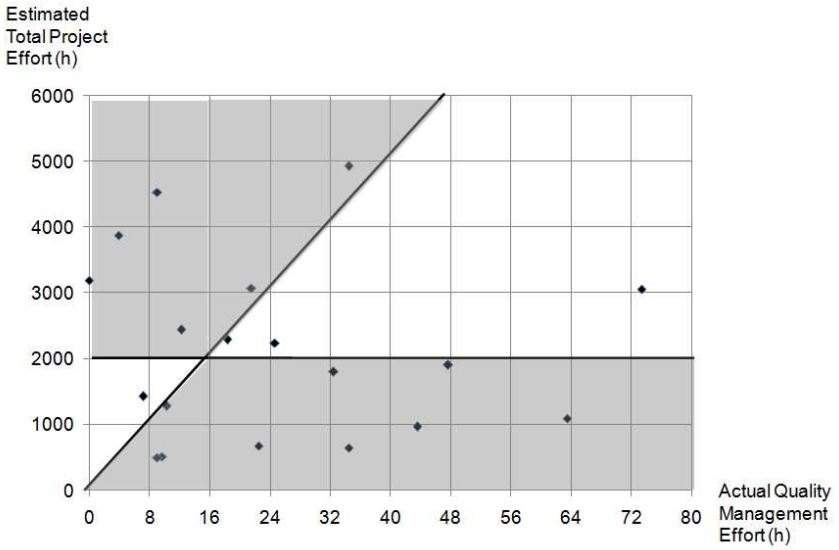

Figure 7. *The theory of actual quality management effort relation to the estimated total project effort*

In the light of our data mining experiment, QM was the only non-construction activity that showed a pattern of estimated total project effort influencing actual effort. One reason for this might be that QM (or its sub-activities) is intuitively considered important by the project team, and is performed whenever possible (when a team's energy is not consumed by software construction). The other supporting activities (e.g., configuration management, orientation, and documentation) do not present the same kind of behavior, i.e., no pattern could be found in our data. However, on average in the projects, all the other non-construction activities consumed more effort than QM, which indicates that they are necessary in a software project (cf. (Haapio, 2006)). Moreover, these activities are relevant for the success of the delivery in terms of both software and project quality, and thus should not be considered a cost overhead. Devoting effort to non-construction activities can indeed reduce the effort needed for software construction or project management, and reduce the cost of poor quality in terms of, for example, warranty costs. Hence, all non-construction activities must be carefully considered in software project effort estimation, but unfortunately our data does not reveal to what extent, except for the QM effort.

4.2.3 Improving effort data quality

In Paper III, we examine the adoption of general project activities to increase the reliability of registered effort and uniformity of the work breakdown structure in order to increase effort data quality.

The effort registration on old, familiar time-booking entries can begin immediately, and effort is usually registered on the correct entry. The new project activities and new sets of activities, however, require an adoption period before effort can be registered on the new time-booking entries. The adoption time varies but can at worst last whole project execution which results in poor effort data quality and skewed effort data as effort is registered on wrong activities or is not registered at all. The correct registration of effort is essential for effort monitoring

and effort re-estimations, since from this point on the project's own registered effort is the primary data for re-estimations. The re-estimations benefit also from effort assessments, which are conducted as each project phase, iteration or sprint ends.

In the case study presented in Paper III, a two-phased questionnaire survey was conducted among a large project team totaling 33 persons to explore the efficient adoption of a specific set of software project activities: the non-construction activities. This WBS was new to the project team.

The results include factors that both promote and discourage the adoption of new project activities, and a suggestion for an adjusted work breakdown structure of the non-construction activities.

The main findings for efficient new project activity adoptions include recommendations for versatile and frequent information on both new and old activities and by several sources, emphasized in the beginning of the project. Recommended sources of information include written guidelines sent by e-mails and verbal information given by management. It is notable that only a small minority of choices for information sources are considered somewhat ”poor”. Peer information, or self-initiated questions are considered as ”poor” sources.

Furthermore, it can be beneficial to consider an optimal number of activities for a project. This consideration involves the funneling of effort on correct activity, since leaving project activities out of the WBS increases the probability of misregistration, i.e., effort is registered on a wrong time-booking entry or left unregistered, which skews the effort data. From a project team member's view, the project should contain as small a number of different project activities (time-booking entries) as possible. Moreover, project activity views provided by the effort time-booking system should preferably be customized for the project team member, i.e., only those entry alternatives are shown in the work time booking system's user interface that are necessary for a particular person. The time-booking entries should be titled clearly and consistently between the projects to

achieve consistency. Therefore, the use of mandatory and optional project activity set templates is advantageous.

From the project aspect, the greatest difficulties are caused by naming the project activities as time-booking entries in the work time booking system in project start-up. To benefit effort monitoring and analysis, the entries are named with somewhat cryptic coded titles instead of using long and clear titles, which would have ensure better adoption and understanding. From the information point of view, the new project activities are introduced with an induction e-mail from the project manager regarding the project activities and with an internal project kick-off event where these activities are also explained to the project team. Although according to the project team members these informative actions are appropriate and recommended, a more detailed description and examples of the activities and the registration of effort on them would be endeavored. Moreover, a documented guideline should be included in the project network folder in the future projects. This guideline explains the titles and purposes of the time-booking entries.

Getting the project team to be self-steering with the new project activities is a challenge. Although the information sources promoting self-steering are favored, the high ratios in active information both as favored information sources and the motivating and assisting factors give a clear statement. Furthermore, the project case study results reveal that without a strong commitment on promoting the adoption leaves the project team puzzled about the activities and registrations.

The main findings concerning the WBS of the non-construction activities include that every project should contain a set of mandatory non-construction activities as time-booking entries for effort monitoring, analyzing, and estimation purposes. The non-construction activities provided in the project's WBS that are perceived useful as individual time-booking entries by the majority of the project team members is presented in Table 4.

Table 4. *The non-construction activities of the case project perceived useful as independent time-booking entries by project team members (N=33)*

Activity	**Perceived useful (%)**
Documentation	92.3
Orientation	92.3
Reviews	84.6
Quality assurance	76.9
Customer support	69.2
Project events	61.5
Technical environment maintenance	61.5
Technical environment set-up	53.8

None of the non-construction activities included is considered more useless than useful by the majority. According to the study of Paper III, it appears that a suitable set of non-construction activities includes customer support (including customer support, queries, and training), documentation (non-related to software construction, e.g., user guides), orientation, project team working (including project start-up, events and project-related tasks), quality assurance, reviews, and configuration management (including setup and maintenance of technical environment, configuration and version management).

4.2.4 Improving effort assessment

In Paper IV, a process (Figure 8) and a novel stepwise method is proposed for assessing software project effort. The method provides necessary tools for the effort assessment and, when utilized for post-mortem analysis, for deciding on the project result and how the project differs from the other projects. The deviating projects require different set of analysis questions. The effort assessment method is advantageous to be used for individual projects and a set of multiple projects. A set of projects can be analyzed for the annual projects' report, for instance. A comparison between projects is useful when dealing

with annual projects' reports. The annual projects' report is usually a summary of final reports of projects.

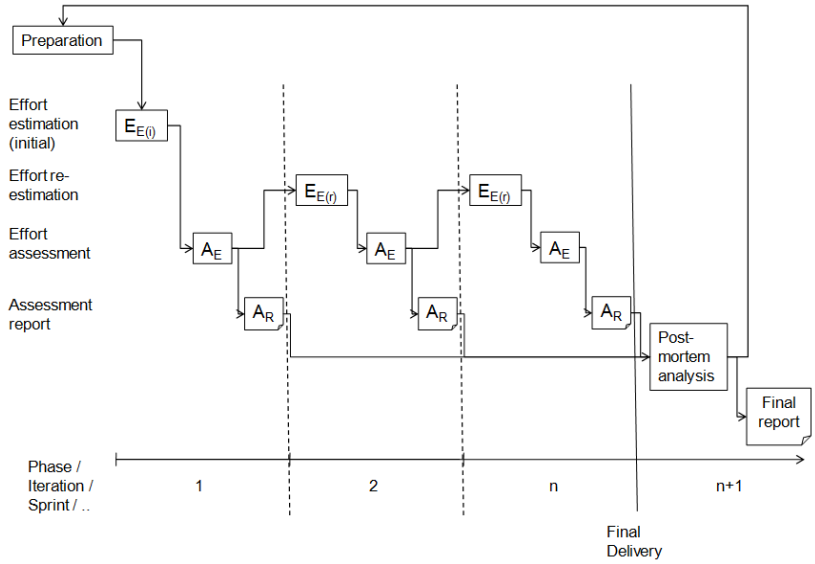

Figure 8. *Effort assessment process*

The project (or projects in the case of project comparison) can be positioned into a coordinate to be assessed in respect to quantity (functional and non-functional) results, quality results, and effort (Figure 9). Here we assume that it is possible to approximate results of the project in relation to functional (quantity) and quality requirements by utilizing discrete values on axes x and y: below, meets, and exceeds. This assessment results in 3x3x3=27 different situations the project can be in. A project meeting the quality and quantity requirements is most favorable in long term customer relationship. A project resulting below requirements leads to customer dissatisfaction and, in some cases, to project cancellation. In cases where a project exceeds its requirements the customer is satisfied but the supplier makes losses.

On axis z, the project effort is considered in respect to its estimation. The effort can be underestimated, accurately estimated, or overestimated. Project effort may be overestimated which may indicate the stakeholder's dissatisfaction though

increased costs. The underestimation indicates that there are some problems to accomplish the project on time and budget, and thus the quantity or quality of results may be decreased.

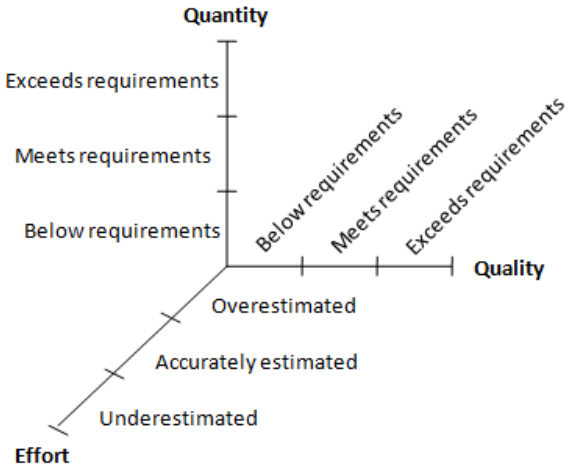

Figure 9. *Project positioning in respect to quantity, quality, and effort*

The stepwise effort assessment method is used for analyzing the effort of a specific set of project activities and their effort estimates compared with those of other projects. Before employing the method the project activity set in question has to be specified and composed. Any activity set can be specified and analyzed with the proposed method.

Analysis of the activity sets of the project is advantageous to discover the relationships between different activity sets that are not isolated and affect each other. Relationships between activity sets may highlight the contradictions between them. Hence, we propose that the relationships between activity sets and their effort should be emphasized, too. It is possible that investing too little effort in one activity set increases the effort of another. We argue this with two simple examples:

- The effort of the activity set 'testing' may have a relationship to the effort of the activity set 'deployment and introduction'. If the components and their interfaces have been

tested and the conformance tested thoroughly, the effort of introduction decreases.

- The activity sets 'architecture design' and 'software implementation' affect each other in the same way. Emphasizing architecture designs facilitates finding design patterns that can be used several times in software construction decreasing the implementation effort. The above examples illustrate that software process improvement initiatives may come forward while the relationships between activity sets and their effort are compared.

One condition for employing the analysis method is that an effort repository has been built. The *effort repository*, a subset of project knowledge repository, contains effort data of the completed projects. The project activities and their effort are collected to the repository, and the activities are reorganized consistently into the repository. These activities can be organized into specific default activity sets, which represent a typical set of activities to be analyzed with the method. The activities can be chosen for the set based on their frequency and significance in projects. Hence, a certain granularity level of the project activities is required. The granularity level defines on how detailed level the activities are broken into. The granularity level is set to support the analysis, effort reporting, and the data collection for the future effort estimations.

The proposed effort assessment method has five steps:

1. A preliminary step where the existence of the effort repository and the usability of the default activity set is confirmed (and built, if necessary).
2. Extraction of a specific set of project activities and the calculation of the actual effort of those activities. The extracted set of project activities is then compared with the default activity set to decide the method applicability in that particular case.
3. Calculation of the total project effort estimation error.
4. Setting the project coordinate and placing projects into it.
5. Analysis of the project or projects.

The five steps of the method are described in detail in Paper IV. As a result of these steps the effort repository is updated with possible new activities belonging to the activity set. The effort information results are used for effort estimation method calibration and project assessment, and final and annual reporting.

The stepwise effort assessment method is evaluated and employed in a case study in Paper IV to analyze the effort of the non-construction activities. While a comparison between our method and others is challenging, the feasibility of the proposed method is shown in the case study considering four projects. First and foremost, the method assists in gathering selectively, based on quantitative effort data, qualitative effort information from the informants to produce iteration assessment, project final and annual project portfolio reports. Secondly, the method can be employed to collect calibration input for the effort estimation method or model; the method provides effort calibration information as a result to populate a repository from which the effort estimation method's factor weights can be derived.

4.2.5 Improving effort management process

Paper V introduces a novel framework for software project effort management. The framework includes three perspectives into effort management: effort management lifecycle with different phases, necessary functions during those phases, and software project activities (Figure 10). These perspectives influence on each other.

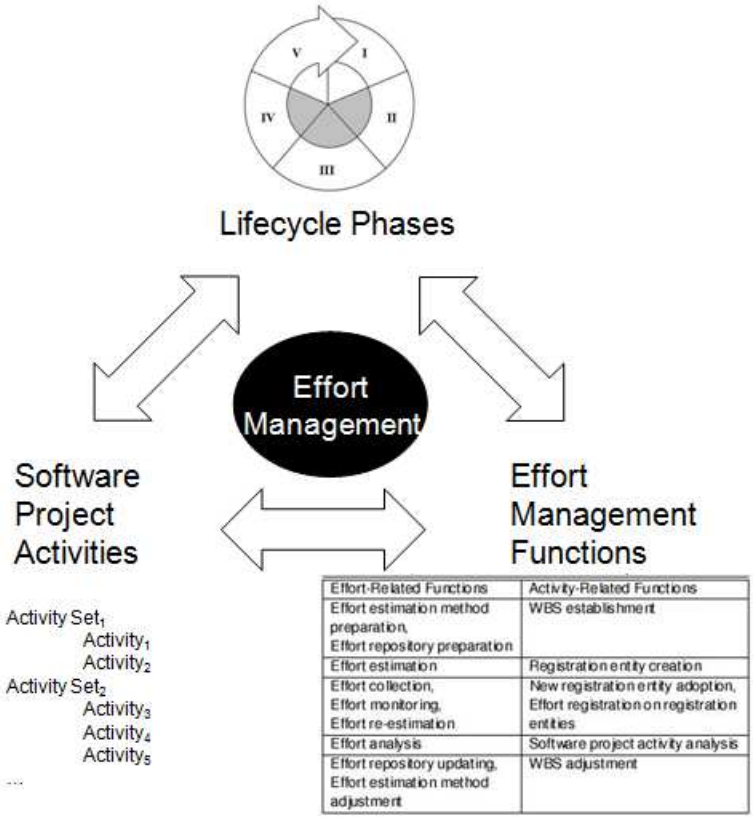

Figure 10. *The three perspectives of the effort management framework*

The effort management lifecycle can be divided into continuous five phases: a phase preceding software project's three phases, three project-related middle phases (pre-project, project, and post-project) referring to software project's lifecycle (phases II-IV highlighted with gray in Figure 10), and a phase following the software project's phases.

Effort management starts before a project exists. The effort management functions include the establishment of a general, project-independent WBS. Also, the effort estimation method (or, tool) for the software project is either acquired or a proprietary estimation method is constructed, which after the method is initially prepared for estimations, for example with effort or cost driver calibration.

In project's lifecycle, during the pre-project planning, the project is planned and set up. Project planning includes the project activity planning and effort estimation sub-functions. Effort is initially estimated with the method in use with project information supplied by the customer. At this point, the different activities concerning the project are also planned. These activities include the activities related to actual software construction, project management, and other project-related activities. The planned activities are created during the project setup into the work time booking system as time-booking entries for effort registrations during the project execution. These entries are organized into particular activity sets as a work breakdown structure.

During the actual project phase the project effort is collected, monitored, assessed, and re-estimated. Effort is collected during the project execution to monitor its realization. If actual effort deviates from the planned, the reasons are explored and necessary actions are taken. Furthermore, effort is collected for re-estimations with adjusting project-specific data, and for effort analysis after the project has been completed. The effort collection includes effort registration by the project team members on time-booking entries in the work time booking system. The new project activities and new sets of activities require an adoption period before effort can be registered on the new entries.

During project closure, the post-project phase, the delivered project is analyzed, and a project final report is drawn. As a sub-function, effort analysis is conducted to produce input for a thorough, usually qualitative, project post-mortem analysis. In effort analysis the actual effort of project activities are compared with the estimated, and the reasons for the accuracy or inaccuracy are analyzed and explained. Effort analysis produces effort information for improving and calibrating the estimation method, thus improving the software process. Moreover, if new significant activities are identified, they are recorded for activity planning of future projects.

In our framework, effort management ends into a phase following a project's lifecycle. The estimation method is adjusted after the project has been post-mortem analyzed. The analyzed effort data is stored in the effort repository. The effort repository requires preparation and adjusting actions during the first and last phase, respectively.

Besides effort management lifecycle, Paper V presents the other framework elements, software project activities and effort management functions (see also Table 3 in sub-chapter 4.2.1), in detail.

4.3 EVALUATION OF THE RESULTS

In this sub-chapter, we first evaluate the research results of each paper against those four perspectives that are applicable for the paper in question, and then discuss the limitation of the study. Our four evaluation perspectives are:

1. how the results address the research problems,
2. research artifacts' comparison with prior alternative solutions,
3. how the results meet the research evaluation criteria for constructive research artifacts, and
4. the level of deployment of the research results at the research site.

4.3.1 Addressing research problems

In sub-chapter 3.1, we stated six research problems we aim to provide improvements as solutions. In Table 5, we map the research problems with the research papers, i.e., whether the research results of the research papers address the research problems either directly (d) or indirectly (i).

Table 5. *Research papers addressing directly (d) or indirectly (i) to the research problems*

Research problem	**Research paper**				
	I	**II**	**III**	**IV**	**V**
RP1: *"Effort (and cost) estimates are inaccurate."*	i	d	i	i	i
RP2: *"Estimators consider mostly core construction activities, relevant in their work, when estimating."*	i	d	i	i	d
RP3: *"Effort data quality is bad, i.e., it is not reliable."*	i		d	i	i
RP4: *"Effort data is seldom assessed."*	i		i	d	i
RP5: *"Effort-related functions are presented in a scattered way."*	i				d
RP6: *"The effort estimation models, methods and applications are 'black boxes', i.e., the estimator cannot judge the result."*		d		i	

Paper I addresses indirectly to all but one of our research problems by providing a framework for improving effort management. The utilization of the framework aims to increase effort estimates, and to increase knowledge of the different activities and sets of activities related to a software project. Effort collection and assessment are included in the framework as effort-related functions. The effort-related functions are assembled together in one framework.

Paper II with the findings of non-construction activities and their effort addresses directly three research problems (RP1, RP2 and RP6). The research results of Paper II include both knowledge on software project activities and related effort. Thus they promote the transparency of those activities in estimation models and applications.

Paper III with the findings to improve the adoption of new software project activities addresses directly to research problem 3 since the results can be used in improving the quality (and thus reliability) of effort data. Thus Paper III indirectly

addresses to research problems related to effort estimation accuracy (RP1), consideration of different activities in estimates (RP2), and the assessment of effort data (RP4).

Paper IV addresses directly to research problem 4 by providing a process and novel method to assess software project effort. Indirectly, Paper IV addresses the problems related to estimation accuracy (RP1), activity consideration (RP2), effort data quality (RP3), and the transparency of software project activities and related effort (RP6), as the method can be used to reveal such information.

Paper V with the effort management framework addresses directly to two research problems (RP2 and RP5) by having software project activities as one of the three perspectives in the framework, and assembling effort-related functions together within the framework. Paper V also addresses indirectly to three research problems (RP1, RP3 and RP4) as the utilization of the framework aims to increase effort estimates, and as the effort collection and assessment are included in the framework as effort-related functions.

4.3.2 Comparisons to alternative solutions

A novel research artifact must provide "significant improvement" (Järvinen, 2001; March and Smith, 1995) compared to previous ones. A comparison to prior solutions is possible for the effort management frameworks (Paper I and V), and for the effort assessment method (Paper IV). Comparisons to the alternative solutions are, however, complicated by their differences, e.g., purpose.

A comparison between the frameworks proposed in papers I and V and the existing frameworks and capability maturity models implies that some major elements related to effort management seem to have been previously neglected (Table 6):

- A lifecycle related to effort management is identified in our frameworks and in the SDM framework (Tsoi, 1999). However, the SDM framework is limited mostly to the pre-project and project phases whereas our framework includes phases before and after these

two phases. The lifecycle the assessment models refer to is the process to be evaluated.

- All examined frameworks and assessment models identify several effort-related functions. The frameworks, however, concentrate mostly on the effort estimation activity whereas our frameworks identify a larger range of effort-related functions concerning a software project.
- The role of software project activities is acknowledged in all but two frameworks (Tsoi, 1999; Vesterinen, 1998). However, effort distribution is not included in the other frameworks as a key perspective into effort management whereas the activities and sets of activities are a key perspective for our frameworks. Only our frameworks include activity-related functions. The assessment models identify activity-related functions, but only ISO/IEC 15504 (ISO, 1993) specifies them in more details.

The process maturity assessment models do not provide direct tools to improve effort management rather than requirements for the software engineering process to consider effort management, i.e., the models tell 'what' to do (or should be done) rather than 'how' it should be done. The assessment models are focused on the improvement of the technical processes of software development rather than project management, in which effort management commonly is included in the assessment models.

Table 6. *A comparison of effort management related frameworks on three effort management perspectives*

Framework	**Identifies effort management lifecycle phases**	**Specifies effort management functions: effort- and activity-related**	**Acknowledges software project activities**
(Fairley, 1992)	None	Effort-related	Yes
SDM (Tsoi, 1999)	2 (emphasizes forepart)	Effort-related	No
(Vesterinen, 1998)	3 (process improvement related)	Effort-related (process improvement related)	No
CMM's (Paulk, et al., 1993)	None	Both (identifies, does not specify)	Yes (does not specify)
ISO/IEC 15504's (ISO, 1993)	None	Both	Yes
CMMI's (SEI, 2006)	None	Both (identifies, does not specify)	Yes (does not specify)
Framework for improving effort management (Paper I)	5	Both	Yes
Effort management framework (Paper V)	5	Both	Yes

Our frameworks aim to improve the accuracy of effort estimations. Therefore, the effort-related functions which are influenced by effort but which do not (significantly) influence effort are delimited from the frameworks. An example of such a function is resource allocation, which has an influence on the effort estimation, since resources are not identical, i.e., other members are more experienced than others and thus require less effort to complete a task. However, we consider this influence to be rather insignificant, since project teams are usually

heterogenic, i.e., the team consists of both more- and less-experienced members.

In general, the postmortem analysis presentations describe capturing the attributes of the whole project, whereas the effort assessment method proposed in Paper IV focuses on capturing the effort information related to a project or projects in a focused and thus efficient manner. To our best knowledge, such a detailed method is not presented so far in the context of effort assessment.

Most postmortem analysis presentations introduced in sub-chapter 2.2.3 (Birk, et al., 2002; Brady and DeMarco, 1994; Collier, et al., 1996; Desouza, et al., 2005a, 2005b; Dingsøyr, et al., 2001; Nolan, 1999; Tiedeman, 1990) refer only vaguely to software project effort (or cost), mostly as examples of project effort, cost or productivity attributes that need to be either collected or assessed. One third (i.e., five) of the papers do not refer to effort or cost at all (Anquetil, et al., 2007; Dingsøyr, 2005; Kinoshita, 2008; Kumar, 1990; Maham, 2008). The sole method-oriented paper (Sertic, et al., 2007) presents only one general question on effort when assessing team efficiency.

Out of 15 presentations prior to ours, only Kerth's (1998, 2001) describe in more detail how to collect effort information. He presents the collection of effort information as questions to be stated in interviews (postmortem meetings or email queries). The example questions presented address the same relevant information that have an influence on each other (effort, and quantity and quality results) as our approach. However, we attempt to provide a more systematic approach to assess project effort with our method than that of Kerth. We promote a self-steered assessment by providing a detailed and stepwise process and method where the effort can be assessed by the project manager and team without an external facilitator. Facilitator-based approaches are challenging for large companies due to distributed software engineering and the large number of projects needed to be assessed in a short time.

Furthermore, in effort assessment the same question set does not apply in every situation. For example, factors, such as actual

effort and the estimate error, affect the consideration. Hence, it is advantageous to analyze project activities in separate sets, which in turn have to be grouped and equipped with questions in a reasonable way.

4.3.3 Research evaluation criteria for constructive research artifacts

We can identify two main types of constructive research products in our research: model and method. We apply the research evaluation criteria (Table 2) for a model in evaluating the two frameworks (Papers I and V), and the research evaluation criteria for a method in evaluating the effort assessment method (Paper IV).

The criteria metrics for a model include fidelity with real world phenomena, completeness, level of detail, robustness, and internal consistency (March and Smith, 1995), and form and content (Järvinen, 2001).

The context of effort management is quite universal, i.e., different organizations with software projects follow more or less same process and perform same effort-related functions. A model (here; the framework) is an abstraction of the real world, and completeness of the model in relation to the reality cannot be demanded (Järvinen, 2001). Nevertheless, these criteria were the driving forces in building the proposed frameworks. Besides completeness and a suitable level of detail, robustness was a key premise for the frameworks. One research problem was the earlier fragmented presentations of effort management, especially in the case of capability maturity models. The criterion on internal consistency is a natural requirement from the research point of view (cf. (Järvinen, 2001)). To support the criterion for form and content, special consideration is given on the presentation to support communication and diminish misunderstanding. The constructed innovation must provide significant improvement (Järvinen, 2001; March and Smith, 1995). The proposed effort management frameworks were constructed based on the case study analysis on three other frameworks (Fairley, 1992; Tsoi, 1999; Vesterinen, 1998) and

three process maturity assessment models (ISO, 1993; SEI, 1993; SEI, 2006).

The criteria metrics for method evaluation include operationality, efficiency, generality, and ease of use. In other words, the research results need to be *pragmatic*. Practicality is especially considered throughout our research. We evaluate our effort assessment method as suggested in (Järvinen, 2001): by showing its feasibility in practical use. Our research originates from practical demands, i.e., we study topics yearning improvement. The data sample was based on real projects, and results were evaluated with prototyping in Tieto Finland Oy, and the results were reported as case studies.

Although the results were evaluated in case studies with the non-construction activities, the results could have been evaluated with other activity sets as well. The research results can be generalized to consider all software project activities. Moreover, our research is not limited to just only one activity set and only one phase. Instead, our studies also include crossing the different activity sets and phases.

4.3.4 Deployment of the research results

In this sub-chapter, we discuss how the research results are deployed at the research site.

Most of the research artifacts presented in this thesis have been gradually integrated into Tieto Corporation's group-level project management process as a part of Tieto's software process improvement activities. The artifacts have been piloted with real business cases either on unit or group-level in Tieto.

By the end of 2010, the main concepts of the research were introduced in process material, i.e., process descriptions, tools, document templates and their guidelines, and project management process training materials. The introduced concepts include both effort management and the work breakdown structure of the non-construction activities.

The process descriptions and document templates acknowledge better the effort management process and its functions, effort assessments and analysis, for example. The

process activities have been complemented with effort assessments and project's post-mortem effort analysis. At the research site, the WBS template for the general software project activities was enhanced with the missing non-construction activities which were identified in our research. The WBS template is utilized in project planning (effort estimation and scheduling), and in effort collection as an example WBS for project's WBS in the work time booking system. Moreover, the finding of the significance and the amount of the non-construction activity effort was included into the planning guidelines.

The research results have been in pilot use in parts of the Tieto Corporation. Here, we provide two examples: effort management framework and effort assessment process and method.

The effort management framework has been employed in software development projects at Telecom & Media, the largest business area within Tieto. As a result of the pilot it was found out that not only the framework increased the project managers' knowledge on effort management related issues, but also improved the adherence to the effort management related process and functions. According to the framework, we distinguished five phases and applied the effort-related and activity-related functions related to each phase. For effort management purposes and to increase the project comparability we divided effort in three major project activity sets: software construction, project management, and non-construction activities, and established a WBS with a generic part concerning the non-construction activities to be employed with software projects.

The deployment of the effort assessment process has taken place gradually at the research site. The foundation of postmortem analyses and final reports were integrated into Tieto's project management process in late 1990s from which on the pre-defined process and document template have been utilized. The process acknowledges and requires an analysis of the effort and outcome of the project. From a reactive approach

of analyzing projects at the end the project, the process evolved to a more proactive approach in mid-2000s when the waterfall development model was complemented with the incremental, iterative, and agile models, which require assessments after each phase, iteration or sprint.

The challenge has been the inconsistent implementation of the assessments and postmortem analyses–the depth of the analysis is based rather on the project manager than the process, and what the manager emphasizes and considers important. The conclusions have remained disconnected, and the dependencies between software project activities and their effort have not been considered.

In analyses, the quantitative part of the method has been applied, i.e., the project activities' actual effort has been recorded with their estimation errors. In practice, however, only the significantly underestimated projects and activities have been analyzed thoroughly. We argue that by adding little more effort (i.e., cost) to how effort assessments are currently performed at Tieto the benefits of the assessments can be significantly increased. Indeed, the value of an assessment limited only to quantitative results can be questioned. Without an extensive qualitative analysis of the project activities and their effort, the assessment results (i.e., benefits) remain moderate, and the assessments do not initiate estimation or software process improvement actions.

Method's utilization is required to reveal a poor estimation. A project-level estimation error does not imply how successful the estimation has actually been. By applying the method we are able to reveal that the accuracy of total project effort is based rather on sheer luck than a thoroughly successfully estimated project.

The relevant group-level project management sub-processes are currently revised to address better the shortcomings. While the awareness of the role of activity sets in effort assessments has increased, the approach has to be deployed throughout the organization. To promote and ease both adaption and the utilization of the method, we are automating the quantitative

part of the method, with an error coordination presentation. This application can state the relevant questions to activity set under examination.

The gathering and calculation of the effort data is very time-consuming if it is not automated. However, the data is very useful in assisting the estimation of forthcoming projects, if the data is feasibly organized. Moreover, a structured data employing default activity sets ensures the comparability of the projects.

4.3.5 Limitations of the study

This thesis suffers from four general limitations related to both the data and methodology. The first concern related to the data is the reliability of the actual effort. As project team members feed their own work hours into the work time booking system, there can be individual variations with the inputs. Moreover, project team members can be too embarrassed to book all the non-construction activity time (e.g., orientation, problem solving, planning, waiting). On the other hand, they may dump all unclear or non-billable project effort on some non-construction activity. Furthermore, the work hours are booked with half-hour precision, which also skews the data. Also, Hochstein, et al., (2005) note that effort recorded varies significantly depending on if it is automated (instrumented) or self-recorded. Hence, self-recorded effort is never quite reliable. However, as these are universal problems with practical effort data, this is accepted for this research. To get near-to-total reliable effort information, the procedure for booking work hours as well as the effort activities should be unified in the organization. Moreover, the project team must be committed to find the correct time-booking entry and record effort on it.

The second concern relates to the data is that they are provided by only one company. However, since the project settings, methods, and staffing varied between teams and the 32 projects, the data sample represents heterogenic project environment.

The third concern relates to the generalization of the results of the studies. As the case studies in the research papers deal with a limited number of cases, they lack statistical significance. This is very typical to qualitative studies within software engineering. Kitchenham, et al., (1995, p. 53) point out that case studies are ”difficult to generalize” and a case study ”cannot be generalized to every possible situation”. Moreover, our case study data set did not contain projects applying agile software development approach, and it can be challenging to apply the results with management approaches that are based on features instead of tasks (Cohn, 2008). Keeping these flaws in mind we conclude with results and promote a cautious approach with the results.

Moreover, we apply case studies merely from the exploratory basis while we seek research problems in building solutions. Hence, the outcomes of this study are emphasized to exploratory rather than explanatory results (cf. (Yin, 1994)).

4.4 CONTRIBUTION OF THE THESIS

The key contribution of the study is the development and validation of artifacts for pragmatic effort management. The outcomes of and the findings for the described effort-related functions can be used in improving the effort estimation accuracy of software projects. However, these outcomes and findings are as important for software project management by themselves.

This study aggregates the fragmented concepts related to effort management. Chapter 2 presents and defines the key effort-related concepts. Two frameworks to be utilized in effort management are introduced in Chapter 4: a research and SPI framework for improving effort management and another framework for managing the effort in software projects. Moreover, proposals for improving effort-related functions in order to increase effort estimation accuracy are presented in Chapter 4. First, the effort of the software project activities, the

non-construction activities in particular, and its impact on software project effort estimation is examined, and a predictor for estimating total software project effort is presented. Second, factors promoting an efficient new project activity adoption are proposed. An efficient adoption of new activities is essential to ensure valid effort data for further assessments and re-estimations. Third, a stepwise effort assessment method is proposed for analyzing effort in a focused manner. The analyzed effort information can be used to validate and calibrate the employed effort estimation method, and to produce assessments, and final and annual project reports.

Our approach on effort management and its three improvement proposals are novel. Although there has been a vast interest in the concepts of software process improvement and effort in software projects within software engineering, researchers have, by and large, overlooked a more complete approach on effort management. Effort has not been seen as an independent area of management like risk or quality. The software engineering literature has focused on software construction and project management, whilst neglecting the third significant group of project activities, the general non-construction project activities, as well as ignored the adoption mechanisms of new project activities. The prior processes for effort assessments provide general guidelines without a detailed method to analyze effort. We, on the other hand, propose a concrete method to analyze software project effort.

This research provided valuable lessons for Tieto Finland Oy regarding effort management. Furthermore, we believe that the results are extendable to software industry in general in the context of software development projects. Moreover, some results can be in fact applicable in a broader context, i.e., in projects in general, or in cases related to effort.

5 Conclusions

A software supplier organization strives to estimate the effort needed in building software as accurately as possible to ensure the project's budget and schedules, and the success of resource allocation. Despite the numerous effort estimation approaches and applications available, the estimates have remained inaccurate. The objective of this study was to improve the management practices of software development project effort, resulting in increased effort estimate accuracy.

Effort has not been seen as a management area of its own in software engineering. The software engineering literature discusses risk management, quality management and configuration management individually but effort is covered as an integrated part of software project management. Also, the consideration has been limited to the actual project phases.

We strive to both broader and more coherent approach on effort management than that in literature. For instance, we have extended the consideration from project's phases to phases before and after project's phases.

In improving the effort management practices, this study commenced in Chapter 2 with the theoretical background and the key concepts and functions related to effort and its management in software development projects. The focus on the theoretical background was largely on effort estimation function since the main research problem in this thesis was to increase estimation accuracy.

In Chapter 3, we presented our research approach with the key research problems and the employed methodologies. The research artifacts were built and evaluated with the constructive research methodology. The main supportive research methodology was case study, which was used both for artifact evaluations and effort data analysis. The data set was comprised

of 32 software development projects, provided by the research site, Tieto Finland Oy.

Chapter 4 took an overview on the research publications. The description of the relation of the research papers to each other continued with the summaries of the research papers proposing improvements on effort management. Each of these improvement proposals relates to phases in the effort management lifecycle, and is included to the framework for effort management. First, we introduced our research framework for improving effort management in software projects. As a part of the framework, we have proposed software project effort to be distributed into three main categories, namely software construction, project management, and non-construction activities. Second, the non-construction activities were presented in more details in a data mining experiment to discover effort predictors to improve effort estimations. Third, the projects' effort data quality was improved by exploring the adoption of the new effort time-booking entries in order to ensure correct work time recording. Fourth, a stepwise effort assessment method was introduced to increase effort accuracy by gaining effort information as a result from applying the method. This effort information can be used to adjust the employed effort estimation method, and to evaluate the estimation method's weights that adjust the factors and drivers which are used to derivate the effort estimates. Fifth, we introduced a framework for effort management in software projects in which each of these three effort-related functions (estimation, collection, and assessment) includes into. Our effort management framework is a process involving a lifecycle of software project with effort-related and activity-related functions. Moreover, the framework contains a two-way view with effort distribution and project phases with relating effort management functions.

Chapter 4 ended with a four-perspective evaluation on the research results how they address the research problems, how well they compare with alternative solutions, how they correspond with the research evaluation criteria in constructive

research, and how the results have been deployed to practice at the research site.

Based on the research evaluation, we can conclude that our six research problems have reached their target states. Once we have gathered experiences from utilizing the proposed improvements, new needs may emerge. As future work, a thorough study on the non-construction activity effort is endeavored to explore if the effort proportions of software construction, project management, non-construction activities can be stabilized within boundaries, which in turn can increase the estimation accuracy. Moreover, the evolving software engineering and industry generates interesting topics for further research. For instance, as agile software development increases its popularity, replicated case studies are needed to show the usefulness and applicability of our results in agile projects which do not utilize effort or tasks in their project planning (Cohn, 2008). In addition, the interest in software engineering industry on off-shoring complements the non-construction activities, in particular, with many supporting activities that are essential in distributed software engineering. Hence, a further study on an enhanced work breakdown structure for the non-construction activities is necessary in producing structure templates to be applied in effort management.

REFERENCES

Abdel-Hamid, T., 1984. *The dynamics of software development project management: an integrative system dynamics perspective*. PhD thesis: Massachusetts Institute of Technology. [Online] Available at: http://dspace.mit.edu/bitstream/handle/1721.1/38235/12536707.pdf?sequence=1. [Accessed: 1 June 2010].

Agarwal, R., Kumar, M., Yogesh, Mallick, S., Bharadwaj, R.M. & Anantwar, D., 2001. Estimating software projects. *ACM SIGSOFT Software Engineering Notes*, 26(4), pp. 60-67.

Albrecht, A.J. & Gaffney, J.E., 1983. Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, 9(6), pp. 639-648.

Anquetil, N., de Oliveira, K.M., de Sousa K.D. & Batista Dias M.G., 2007. Software maintenance seen as a knowledge management issue. *Information and Software Technology*, 49(5), pp. 515-529.

Armour, P., 2002. Ten Unmyths of Project Estimation. *Communications of the ACM*, 45(11), pp. 15-18.

Avison, D., 2002. Action Research: A Research Approach for Cooperative Work. In: *7th International Conference on Computer Supported Cooperative Work in Design*. Rio de Janeiro, Brazil, 25-27 September 2002. IEEE, pp. 19-24.

Avison, D., Lau, F., Myers, M. & Axel Nielsen, P., 1999. Action Research. *Communications of the ACM*, 42(1), pp. 94-97.

Awazu, Y., Desouza, K.C. & Evaristo, J.R., 2004. Stopping Runaway Information Technology Projects. *Business Horizons*, 47(1), pp. 73-80.

Banker, R.D., Kauffman, R.J. & Kumar, R., 1991. An Empirical Test of Object-based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment. *Journal of Management Information Systems*, 8(3), pp. 127-150.

Banker, R.D., Kauffman, R.J., Wright, C.W. & Zweig, D., 1994. Automating Output Size and Reuse Metrics in a Repository-Based Computer-Aided Software Engineering (CASE) Environment. *IEEE Transactions on Software Engineering*, 20(3), pp. 169-187.

Basili V. & Rombach, H., 1988. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering*, 14(6), pp. 758-773.

Baskerville, R.L., 1999. Investigating Information Systems with Action Research. *Communication of the AIS*, 2(19), pp. 1-32.

Baskerville, R.L. & Wood-Harper, A.T., 1996. A critical perspective on action research as a method for information systems research. *Journal of Information Technology*, 11, pp. 235-246.

Birk, A., Dingsøyr, T. & Stålhane, T., 2002. Postmortem: Never Leave A Project without It. *IEEE Software*, 19(3), pp. 43-45.

Blackburn, J.D., Scudder, G.D. & Van Wassenhove, L.N., 1996. Improving Speed and Productivity of Software Development: A Global Survey of Software Developers. *IEEE Transactions on Software Engineering*, 22(12), pp. 875-885.

Boehm, B.W., 1981. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.

Boehm, B., Clark, B., Horowitz, E., Westland, E., Madachy, R. & Selby, R., 1995. Cost Models for Future Life Cycle Processes: COCOMO 2. *Annals of Software Engineering*, 1, pp. 57-94.

Boehm, B., Horowitz, E., Madachy, R., Reifer, D., Clark, B., Steece, B., Brown, A., Chulani, S. & Abts, C., 2000. *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice-Hall.

Brady, S. & DeMarco, T., 1994. Management-Aided Software Engineering. *IEEE Software*, 11(6), pp. 25-32.

Briand L.C., El Emam, K., Surmann, D., Wieczorek, I. & Maxwell, K.D., 1999. An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques. In: *21st International Conference on Software Engineering (ICSE'99)*, Los Angeles, California, USA, 16-22 May 1999. ACM, pp. 313-322.

Briand, L.C., Langley, T. & Wieczorek, I., 2000. A replicated Assessment and Comparison of Common Software Cost Modeling Techniques. In: *22nd International Conference on Software Engineering (ICSE'00)*. Limerick, Ireland, 4-11 June 2000. ACM Press, pp. 377-386.

Brooks, F.P. Jr., 1975. *The Mythical Man-Month. Essays on Software Engineering*. Reading, MA: Addison-Wesley Publishing Company.

Chatters, B., Henderson, P. & Rostron, C., 1999. An Experiment to Improve Cost Estimation and Project Tracking for Software and Systems

Integration Projects. In: *25th Euromicro Conference (EUROMICRO '99)*. Milan, Italy, 8-10 September 1999. IEEE Computer Society, pp. 2177-2184.

Chen, Z., Menzies, T., Port, D. & Boehm, B., 2005. Finding the Right Data for Software Cost Modeling. *IEEE Software*, 22(6), pp. 38-46.

Clarke, B.K., 1997. *The Effects of Software Process Maturity on Software Development Effort*. PhD thesis: University of Southern California.

Cohn, M., 2008. *Agile Estimating and Planning*. Upper Saddle River, NJ: Prentice Hall.

Collier, B., DeMarco, T. & Fearey, P., 1996. A Defined Process for Project Post-Mortem Review. *IEEE Software*, 13(4), pp. 65-72.

Conte, S.D., Dunsmore, H.E. & Shen, V.Y., 1986. *Software Engineering Metrics and Models*. Menlo Park, CA: Benjamin/Cummings Publishing Company.

Desouza, K.C., Dingsøyr, T. & Awazu, Y., 2005a. Experiences with Conducting Project Postmortems: Reports vs. Stories and Practitioner Perspective. In: *38th Annual Hawaii International Conference on System Sciences (HICSS '05)*. Hawaii, USA, 3-6 January 2005, pp. 233c.1-10.

Desouza, K.C., Dingsøyr, T. & Awazu, Y., 2005b. Experiences with conducting project postmortems: reports versus stories. *Software Process: Improvement and Practice*, 10(2), pp. 203-215.

Dingsøyr, T., 2005. Postmortem reviews: purpose and approaches in software engineering. *Information and Software Technology*, 47(5), pp. 293-303.

Dingsøyr, T., Moe, N.B. & Nytrø, Ø., 2001. Augmenting Experience Reports with Lightweight Postmortem Reviews. In: Lecture Notes in Computer Science, vol. 2188, F. Bomarius & S. Komi-Sirviö (eds.), *Third International Conference on Product Focused Software Process Improvement (PROFES 2001)*. Kaiserslautern, Germany, 10-13 September 2001. Springer Verlag: Kaiserslautern, Germany, pp. 167-181.

Dybå, T. & Dingsøyr, T., 2008. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9-10), pp. 833-859.

Fairley, R.E., 1992. Recent Advances in Software Estimation Techniques. In: *14th International Conference on Software Engineering*. Melbourne, Australia, 11-15 May 1992. ACM Press, pp. 382-391.

Fenton, N.E. & Pfleeger, S.L., 1997. *Software Metrics: A Rigorous & Practical Approach*. 2nd ed. Boston, MA: PWS Publishing Company.

Finnie, G.R. & Wittig, G.E., 1996. AI Tools for Software Development Effort Estimation. In: *1996 International Conference on Software Engineering: Education and Practice (SE: E&P '96)*. Dunedin, New Zealand, 24-27 January 1996. IEEE, pp. 346-353.

Fioravanti, F. & Nesi, P., 2001. Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems. *IEEE Transactions on Software Engineering*, 27(12), pp. 1062-1084.

Garmus, D. & Herron, D., 2001. *Function Point Analysis: Measurement Practices for Successful Software Projects*. Upper Saddle River, NJ: Prentice-Hall.

Garrick, J., Chan, A. & Lai, J., 2004. University-industry partnerships. Implications for industrial training, opportunities for new knowledge. *Journal of European Industrial Training*, 28(2/3/4), pp. 329-338.

Glaser B. & Strauss A., 1967. *Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago, IL: Aldine Transaction.

Gray, A.R, MacDonell, S.G. & Shepperd, M.J., 1999. Factors systematically associated with errors in subjective estimates of software development effort: the stability of expert judgment. In: *Sixth International Software Metrics Symposium*. Boca Raton, FL, USA, 4-6 November 1999. IEEE, pp. 216-227.

Gray, C.F. & Larson, E.W., 2006. *Project Management: The Managerial Process*. 3rd ed. Singapore: McGraw-Hill.

Greenwood, D.J. & Levin, M., 1998. *Introduction to Action Research. Social Research for Social Change*. Thousand Oaks, CA: Sage Publications, Inc.

Grimstad, S., Jørgensen, M. & Moløkken-Østvold, K., 2005. Software effort estimation terminology: The tower of Babel. *Information and Software Technology*, 48, pp. 302-310.

Gruschke, T.M. & Jørgensen, M., 2008. The Role of Outcome Feedback in Improving the Uncertainty Assessment of Software Development Effort Estimates. *ACM Transactions on Software Engineering and Methodology*, 17(4), pp. 20.1-34.

Haapio T., 2004. The Effects of Non-Construction Activities on Effort Estimation. In: *27th Information Systems Research in Scandinavia (IRIS'27)*. Falkenberg, Sweden, 14-17 August 2004.

Haapio, T., 2006. Generating a Work Breakdown Structure: A Case Study on the General Software Project Activities. In: International Proceedings Series 6, Tukiainen, M., Messnarz, R., Nevalainen, R. & Koinig, S. (eds.), University of Joensuu, *13th European Conference on European Systems & Software Process Improvement and Innovation (EuroSPI'2006)*, Joensuu, Finland, 11-13 October 2006, pp. 11.1-11.

Haapio, T. & Eerola, A., 2006. Post-Project Effort Analysis Method. In: Vol. I (Organizational Models and Information Systems), M. Cunha & Á. Rocha (eds.), *1st Iberic Conference on Information Systems and Technologies (CISTI)*. Ofir, Portugal, 21-23 June 2006. Portugal: Microsoft, pp. 3-19.

Haapio, T. & Menzies, T., 2009. Data Mining with Software Industry Data: A Case Study. In: Vol. II, H. Weghorn & P. Isaías (eds.), *IADIS International Conference Applied Computing 2009 (AC'2009)*. Rome, Italy, 19-21 November, 2009. IADIS Press, pp. 33-38.

Heijstek, W. & Chaudron, M.R.V., 2008. Evaluating RUP Software Development Processes Through Visualization of Effort Distribution. In: *34th Euromicro Conference Software Engineering and Advanced Applications (SEAA '08)*. Parma, Italy, 3-5 September 2008, pp. 266-273.

Henderson-Sellers, B., 2003. Method Engineering for OO Systems Development. *Communications of the ACM*, 46(10), pp. 73-78.

Herzum P. & Sims O., 2000. *Business Component Factory. A Comprehensive Overview of Component-Based Development for the Enterprise*. New York: Wiley Computer Publishing.

Hevner, A.R., March, S.T., Park, J. & Ram, S., 2004. Design Science in Information Systems Research, *MIS Quarterly*, 28(1), pp. 75-105.

Hihn, J. & Habib-agahi, H., 1991. Cost Estimation of Software Intensive Projects: A Survey of Current Practices. In: *13th International Conference on Software Engineering*. Austin, Texas, USA, 13-16 May 1991. IEEE, pp. 276-287.

Hochstein, L., Basili, V. R., Zelkowitz, M. V., Hollingsworth, J. K. & Carver, J., 2005. Combining self-reported and automatic data to improve programming effort measurement. In: *10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*. Lisbon, Portugal, 5-9 September 2005. ACM, pp. 356-365.

Holmes, G., Donkin, A. & Witten, I.H., 1994. WEKA: A Machine Learning Workbench. In: *1994 Second Australian and New Zealand Conference on Intelligent Information Systems*. Brisbane, Australia, 29 November - 2 December 1994, pp. 357-361.

Iacovou, C.L. & Dexter, A.S., 2005. Surviving IT project cancellations. *Communications of the ACM*, 48(4), pp. 83-86.

Iivari, J., 1991. A Paradigmatic Analysis of Contemporary Schools of IS Development. *European Journal of Information Systems*, 1(4), pp. 249-272.

ISO, 1993. *ISO/IEC TR2 15504, Part 1 - Part 9, Information Technology - Software Process Assessment*. Geneva, Switzerland: ISO.

ISO, 1994. *Quality management and quality assurance standards*. Geneva, Switzerland: ISO.

ISO, 1995. *ISO/IEC 12207, Information Technology - Software Life Cycle Processes*. Geneva, Switzerland: ISO.

Jalote, P., 2000. *CMM in Practice: Processes for Executing Software Projects at Infosys*. Reading, MA: Addison-Wesley.

Jones, C., 1995. Backfiring: Converting Lines of Code to Function Points. *IEEE Computer*, 28(11), pp. 87-88.

Juristo, N. & Moreno, A., 2001. *Basics of Software Engineering Experimentation*. Boston, MA: Kluwer Academic Publishers.

Järvinen, P., 2001. *On Research Methods*. Tampere, Finland: Opinpajan kirja.

Järvinen, P., 2007. Action Research is Similar to Design Science. *Quality and Quantity*, 41, pp. 37-54.

Jørgensen, M., 2004a. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70, pp. 37-60.

Jørgensen, M., 2004b. Top-down and bottom-up expert estimation of software development effort. *Information and Software Technology*, 46, pp. 3-16.

Jørgensen, M., 2005. The "Magic Step" of Judgment-Based Software Effort Estimation. In: *International Conference on Cognitive Economics*. New Bulgarian University, Sofia, 5-8 August 2005, pp. 105-114.

Jørgensen, M., 2007. Estimation of software development work effort: evidence on expert judgment and formal models. *International Journal of Forecasting*, 3(3), pp. 449-462.

Jørgensen, M. & Sjøberg, D., 2001. Impact of effort estimates on software project work. *Information and Software Technology*, 43(15), pp. 939-948.

Jørgensen, M. & Sjøberg, D.I.K., 2004. The impact of customer expectations on software development effort estimates. *International Journal of Project Management*, 22, pp. 317-325.

Kemerer, C.F., 1987. An Empirical Validation of Software Cost Estimation Models. *Communications of the ACM*, 30(5), pp. 416-429.

Kemerer, C.F. & Porter, B.S., 1992. Improving the Reliability of Function Point Measurement: An Empirical Study. *IEEE Transactions on Software Engineering*, 18(11), pp. 1011-1024.

Kemmis, S. & McTaggart, R., 2005. Participatory Action Research: Communicative Action and the Public Sphere. In: N.K. Denzin & Y.S. Lincoln (eds.) 2005. *The Sage Handbook of Qualitative Research*. 3rd ed. Thousand Oaks, CA: Sage Publications, Inc. pp. 559-604.

Kerth, N.L., 1998. *An approach to postmorta, postparta and post project review*. [Online] Available at: http://c2.com/doc/ppm.pdf. [Accessed: 4 August 2010].

Kerth, N.L., 2001. *Project Retrospectives: A Handbook for Team Reviews*. New York: Dorset House Publishing.

Kinoshita, F., 2008. Practices of an Agile Team. In: G. Melnik, P. Kruchten & M. Poppendieck (eds.), *Agile Conference (AGILE '08)*. Toronto, Ontario, Canada, 4-8 August 2008. IEEE Computer Society, pp. 373-377.

Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J. & Linkman, S., 2009. Systematic literature reviews in software engineering - A systematic literature review. *Information and Software Technology*, 51, pp. 7-15.

Kitchenham, B., Pfleeger, S.L., McColl, B. & Eagan, S., 2002. An empirical study of maintenance and development estimation accuracy. *Journal of Systems and Software*, 64(1), pp. 57-77.

Kitchenham, B., Pickard, L. & Pfleeger, S.L., 1995. Case Studies for Method and Tool Evaluation. *IEEE Software*, 12(4), pp. 52-62.

Klein, H. K. & Myers, M. D., 1999. A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, 23(1), pp. 67-93.

Koivisto, M., 2007. Development of Quality Expectations in Mobile Information Systems. In: *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering*. University of Bridgeport, CT, USA, 3-12 December 2007. Springer, pp. 336-341.

Kumar, K., 1990. Post Implementation Evaluation of Computer-Based Information Systems: Current Practices. *Communications of the ACM*, 33(2), pp. 203-212.

Lee, A., Green, B. & Brennan, M., 2000. Organisational knowledge, professional practice and professional doctorate at work. In: J. Garrick & Rhodes, C. (eds.). 2000. *Research and Knowledge at Work. Perspectives, case-studies and innovative strategies*. London, UK: Routledge. Pp. 117-136.

Lyytinen, K. & Robey, D., 1999. Learning Failure in Information Systems Development. *Information Systems Journal*, 9(2), pp. 85-101.

MacDonell, S.G. & Shepperd, M.J., 2003. Using Prior-Phase Effort Records for Re-estimation During Software Projects. In: *Ninth International Software Metrics Symposium (METRICS'03)*. Sydney, Australia, 3-5 September 2003. IEEE Computer Society, pp. 1-13.

Maham, M., 2008. Planning and Facilitating Release Retrospectives. In: G. Melnik, P. Kruchten & M. Poppendieck (eds.), *Agile Conference (AGILE '08)*. Toronto, Ontario, Canada, 4-8 August 2008. IEEE Computer Society, pp. 176-180.

March, S.T. & Smith, G.F., 1995. Design and Natural Science Research on Information Technology. *Decision Support Systems*, 15(4), pp. 251-266.

McBride, T., Henderson-Sellers, B. & Zowghi, D., 2004. Project management capability levels: An empirical study. In: *11th Asia--Pacific Software Engineering Conference (APSEC'04)*. Washington, DC, USA, 30 November – 3 December 2004. IEEE Computer Society, pp. 56-63.

Menzies, T., Port, D., Chen, Z., Hihn, J. & Stukes, S., 2005. Validation Methods for Calibrating Software Effort Models. In: *27th International Conference on Software Engineering (ICSE'05)*. St. Louis, Missouri, USA, 15-21 May 2005. ACM Press, pp. 587-595.

Milicic, D. & Wohlin, C., 2004. Distribution Patterns of Effort Estimations. In: *30th EUROMICRO Conference (EUROMICRO'04)*. Rennes, France, 1-3 September 2004. IEEE Computer Society, pp. 422-429.

Mohagheghi, P., Anda, B. & Conradi, R., 2005. Effort Estimation of Use Cases for Incremental Large-Scale Software Development. In: *27th International Conference on Software Engineering (ICSE'05)*. St. Louis, Missouri, USA, 15-21 May 2005. ACM Press, 303-311.

Mukhopadhyay, T. & Kekre, S., 1992. Software Effort Models for Early Estimation of Process Control Applications. *IEEE Transactions on Software Engineering*, 18(10), pp. 915-924.

Niessink, F. & van Vliet, H., 1997. Predicting Maintenance Effort with Function Points. In: *1997 International Conference on Software Maintenance (ICSM '97)*. Bari, 1-3 October 1997. IEEE, pp. 32-39.

Nolan, A.J., 1999. Learning from Success. *IEEE Software*, 16(1), pp. 97-105.

Nunamaker, J.F. Jr. & Chen, M., 1990. Systems Development in Information Systems Research. In: Vol. 3, *23rd Annual Hawaii International Conference on System Sciences*. Hawaii, USA, 2-5 January 1990. pp. 631-640.

Nunamaker, J.F. Jr., Chen, M. & Purdin, T.D.M., 1991. Systems Development in Information Systems Research. *Journal of Management Information Systems*, 7(3), pp. 89-106.

O'Regan, G., 2002. *A Practical Approach to Software Quality*. Secaucus, NJ: Springer-Verlag New York, Inc.

Paulk, M.C., Curtis, B., Chrissis, M.B. & Weber, C.V., 1993. *Capability Maturity Model for Software, Version 1.1. Technical Report, CMU/SEI-93-TR-024, ESC-TR-93-177*. Pittsburgh, PA: CMU/SEI.

Petter, S., Mathiassen, L. & Vaishnavi, V., 2007. Five Keys to Project Knowledge Sharing. *IT Professional*, 9(3), pp. 42-46.

Pressman, R.S., 2005. *Software Engineering: A Practioner's Approach*. 6th ed. New York: McGraw-Hill.

Pyle, D., 1999. *Data Preparation for Data Mining*. San Francisco, CA: Morgan Kaufmann Publishers.

Rask, R., 1992. *Automating Estimation of Software Size During the Requirements Specification Phase - An Application of Albrecht's Function Point Analysis within Structured Methods*. PhD thesis: Joensuu, Finland: University of Joensuu.

Rautianen, K., Lassenius, C., Vähäniitty, J., Pyhäjärvi, M. & Vanhanen, J., 2002. A tentative framework for managing software product development in small companies. In: Vol. 8, *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*. Washington DC, USA, 7-10 January 2002. IEEE Computer Society, pp. 3409-3417.

Robertson, S. & Robertson, J., 1999. *Mastering the Requirements Process*. Harlow, UK: Addison-Wesley.

Royce, W., 1998. *Software Project Management: A Unified Framework*. Reading, MA: Addison-Wesley.

Rus, I. & Lindvall, M., 2002. Knowledge Management in Software Engineering. *IEEE Software*, 19(3), pp. 26-38.

Sawyer, S., 2001. A Market-Based Perspective on Information Systems Development. *Communications of the ACM*, 44(11), pp. 97-102.

Schwaber, K., 1995. Scrum Development Process. In: J. Sutherland, D. Patel, C. Casanave, J. Miller & G. Hollowell (eds.), *Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings*. London: Springer.

SEI, 2006. *CMMI for Development, Version 1.2, CMMI-DEV, V1.2. Technical Report CMU/SEI-2006-TR-008, ESC-TR-2006-008*. Pittsburgh, PA: CMU/SEI.

SEI, 2010. *CMMI for Development, Version 1.3, CMMI-DEV, V1.3. Technical Report CMU/SEI-2010-TR-033, ESC-TR-2010-033*. Pittsburgh, PA: CMU/SEI.

Sertic, H., Marzic, K. & Kalafatic, Z., 2007. A Project Retrospectives Method in Telecom Software Development. In: Ž. Car & M. Kušek (eds.), *9th International Conference on Telecommunications (ConTel 2007)*. Zagreb, Croatia, 13-15 June 2007. IEEE, pp. 109-114.

Shepperd, M.J. & Schofield, C., 1997. Estimating Software Projects Effort Using Analogies. *IEEE Transactions on Software Engineering*, 23(12), pp. 736-743.

Shepperd, M.J., Schofield, C. & Kitchenham, B., 1996. Effort Estimation Using Analogy. In: *Proceedings of the 18th International Conference on Software Engineering (ICSE--18)*. Berlin, Germany, 25-29 March 1996. IEEE Computer Society Press, pp. 170-178.

Smith, R.K., Hale, J.E. & Parrish, A.S., 2001. An Empirical Study Using Task Assignment Patterns to Improve the Accuracy of Software Effort Estimation. *IEEE Transactions on Software Engineering*, 27(3), pp. 264-271.

Solingen, R. van & Berghout, E., 1999. *The Goal/Question/Metric Method*. London, UK: McGraw-Hill Education.

Sommerville, I., 2001. *Software Engineering*. 6th ed. Harlow, UK: Pearson Education.

Srinivasan, R. & Fisher, D., 1995. Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering*, 21(2), pp. 126-137.

Standish Group, 1999. *Chaos: A Recipe for Success*.

Stringer, E.T., 1999. *Action Research*. 2nd ed. Thousand Oaks, CA: Sage Publications, Inc.

Symons, C.R., 1988. Function Point Analysis: Difficulties and Improvements. *IEEE Transactions on Software Engineering*, 14(1), pp. 2-11.

Szyperski C., 1999. *Component Software: Beyond Object-Oriented Programming*. Harlow, UK: ACM Press.

Takeuchi, H. & Nonaka, I., 1986. The new new product development game. *Harvard Business Review*, 64(1), pp. 137-146.

Tiedeman, M.J., 1990. Postmortems – Methodology and Experiences. *IEEE Journal on Selected Areas in Communications*, 2(8), pp. 176-180.

Tsoi, H.L., 1999. A Framework for Management Software Project Development. In: *1999 ACM symposium on Applied computing (SAC'99)*. San Antonio, Texas, USA, 28 February – 2 March 1999. ACM Press, pp. 593-597.

Verner, J.M. & Evanco, W.M., 2005. In-House Software Development: What Project Management Practices Lead to Success? *IEEE Software*, 22(1), pp. 86-93.

Vesterinen, P., 1998. A framework and process for effort estimation. In: *9th European Software Control and Metrics Conference, 1999 ACM symposium on Applied computing*. Rome, Italy, 27-29 May 1998.

Virtanen, P., 2003. *Measuring and Improving Component-Based Software Development*. PhD thesis: Turku, Finland: University of Turku.

Wilson, D.N. & Sifer, M.J., 1988. Structured Planning: Project Views. *Software Engineering Journal*, 3, pp. 134-140.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B. & Wesslén, A., 2000. *Experimentation in Software Engineering: An Introduction*. Boston, MA: Kluwer Academic Publishers.

Yin, R. K., 1994. *Case Study Research: Design and Methods*. 2nd ed. Thousand Oaks, CA: Sage Publications.

Young, M.R., 1995. A new paradigm for IS development: Effort management. In: *26th Annual Seminar/Symposium*. New Orleans, LO, 16-18 October 1995. Project Management Institute.

Zahran, S., 1998. *Software Process Improvement*. London, UK: Addison-Wesley.

APPENDICES

APPENDIX A: EFFORT DISTRIBUTION IN ANALYZED 32 SOFTWARE PROJECTS

The median, mean, minimum, first quartile, third quartile, maximum, and standard deviation values of the software construction, project management, and non-construction activity effort of total project effort in 32 analyzed software projects are presented in Table A.1.

Table A.1. *The proportion values of the software construction, project management and non-construction activity effort of total project effort (N=32)*

	Software construction	**Project management**	**Non-construction activities**
Median	76.7%	11.3%	11.2%
Mean	74.7%	11.7%	13.6%
Min	52.1%	2.6%	3.2%
Q1	69.6%	8.1%	8.1%
Q3	80.5%	14.9%	20.6%
Max	90.8%	23.5%	31.1%
SD	.0901	.0490	.0768

The median, mean, minimum, first quartile, third quartile, maximum, standard deviation, number, and frequency values (frequency of an activity appearing in a project) of the non-construction activity effort of total project effort in 32 analyzed software projects are presented in Table A.2.

Table A.2. *The proportion and frequency values of the non-construction activity effort of total project effort (N=32)*

	CM	**Cus**	**Doc**	**Ori**	**Pro**	**QM**	**Misc.**
Md	3.3%	1.6%	2.3%	2.1%	3.0%	0.8%	0.9%
Mean	3.3%	3.8%	3.0%	3.6%	3.4%	1.6%	3.1%
Min	0.2%	0.2%	0.1%	0.2%	0.4%	0.0%	0.3%
Q1	1.5%	0.6%	1.4%	1.0%	2.0%	0.5%	0.4%
Q3	4.8%	3.8%	4.0%	3.3%	4.0%	2.3%	6.1%
Max	7.8%	20.5%	8.1%	15.8%	10.2%	5.8%	7.6%
SD	.0223	.0558	.0230	.0424	.0224	.0170	.0312
n	29	17	23	18	23	21	9
f	90.6%	53.1%	71.9%	56.3%	71.9%	65.6%	28.1 %

Note: CM = configuration management, Cus = customer-related activities, Doc = documentation, Ori = orientation, Pro = project-related activities, QM = quality management, Misc = miscellaneous activities.

APPENDIX B: NON-PARAMETRIC EFFORT AND COST ESTIMATION MODELING TECHNIQUES

Table B.1. *Non-parametric effort and cost estimation techniques (applied from Briand, et al., 1999, 2000) with selected publications*

Technique	Selected publications
Artificial intelligence	[2][4]
Artificial neural networks	[2][7][12][17][18]
Case-based reasoning	[9][14][15][18][19]
Inductive learning (or, rule induction)	[1][11][13][14]
CART algorithm and regression trees	[2][4][17][18]
Optimized set reduction	[3][18]
Bayesian analysis	[5][6]
Machine learning	[9][10][14]
Analogy	[4][15][16]
Stepwise regression	[15]
Fuzzy logic	[8][9][10]

[1] de Almeida, M.A., Lounis, H. & Melo, W.L., 1998. An Investigation on the Use of Machine Learned Models for Estimating Correction Costs. In: *IEEE International Conference on Software Engineering*. Kyoto, Japan, 19-25 April 1998, IEEE Computer Society.

[2] Briand L.C., El Emam, K., Surmann, D., Wieczorek, I. & Maxwell, K.D., 1999. An Assessment and Comparison of Common Software Cost Estimation Modeling Techniques. In: *21st International Conference on Software Engineering (ICSE'99)*, Los Angeles, California, USA, 16-22 May 1999. ACM, pp. 313-322.

[3] Briand, L.C., Basili, V.R. & Hetmanski, C.J., 1993. Developing interpretable models with optimized set reduction for identifying high-risk software components. *IEEE Transactions on Software Engineering*, 19(11), pp. 1028-1044.

[4] Briand, L.C., Langley, T. & Wieczorek, I., 2000. A replicated Assessment and Comparison of Common Software Cost Modeling Techniques. In: *22nd International Conference on Software Engineering (ICSE'00)*. Limerick, Ireland, 4-11 June 2000. ACM Press, pp. 377-386.

[5] Chulani, S., Boehm, B. & Steece, B., 1999. Bayesian Analysis of Empirical Software Engineering Cost Models. *IEEE Transactions on Software Engineering*, 25(4), pp. 573-583.

[6] Devnani-Chulani, S., 1999. *Bayesian Analysis of Software Cost and Quality Models*. PhD thesis: University of Southern California.

[7] Gray, A. & MacDonell, S., 1997. A Comparison of Techniques for Developing Predictive Models of Software Metrics. *Information and Software*

Technology, 39, pp. 425-437.

[8] Gray, A.R. & MacDonell, S.G., 1999. Fuzzy Logic for Software Metric Models throughout the Development Lifecycle. In: *18th International Conference of the North American Fuzzy Information Processing Society (NAFIPS)*. New York, USA, 10-12 June 1999. IEEE, pp. 258-262.

[9] Mendes, E., Mosley, N. & Watson, I., 2002a. A Comparison of Case-Based Reasoning Approaches to Web Hypermedia Project Cost Estimation. In: *WWW2002*. Honolulu, Hawaii, USA, 7-11 May 2002. ACM, pp. 272-280.

[10] Mendes, E., Watson, I., Triggs, C., Mosley, N. & Counsell, S., 2002b. A Comparison of Development Effort Estimation Techniques for Web Hypermedia Applications. In: *Eight IEEE Symposium on Software Metrics (METRICS'02)*. Ottawa, Canada, 4-7 June 2002. IEEE Computer Society, pp. 131-140.

[11] Quinlan, J.R., 1996. Learning Decision Tree Classifiers. *ACM Computing Surveys*, 28(1), pp. 71-72.

[12] Schofield, C., 1998. Non-algorithmic effort estimation techniques. *Technical Report ESERG: TR98-01*, Bournemouth University, Department of Computing.

[13] Selby, R.W. & Porter A.A., 1988. Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis. IEEE Transactions on Software Engineering, 14(12), pp. 743-757.

[14] Shepperd, M. & Kadoda, G., 2001. Using Simulation to Evaluate Prediction Techniques. In: *7th International Software Metrics Symposium (METRICS'01)*. London, England, 4-6 April 2001. IEEE, pp. 349-359.

[15] Shepperd, M.J. & Schofield, C., 1997. Estimating Software Projects Effort Using Analogies. *IEEE Transactions on Software Engineering*, 23(12), pp. 736-743.

[16] Shepperd, M.J., Schofield, C. & Kitchenham, B., 1996. Effort Estimation Using Analogy. In: *18th International Conference on Software Engineering (ICSE-18)*. Berlin, Germany, 25-29 March 1996. IEEE Computer Society Press, pp. 170-178.

[17] Srinivasan, R. & Fisher, D., 1995. Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering*, 21(2), pp. 126-137.

[18] Stensrud, E., 2001. Alternative approaches to effort prediction of ERP projects. *Information and Software Technology*, 43, pp. 413-423.

[19] Vicinanza, S., Prietula, M.J. & Mukhopadhyay, T., 1990. Case-Based Reasoning in Software Effort Estimation. In: *11th International Conference on Information Systems*, 1990, pp. 149-158.

Topi Haapio

Improving Effort Management in Software Development Projects

Software supplier organizations strive to estimate the effort needed in building software as accurately as possible to ensure the project's budget, schedule and optimal resource allocation. Despite the numerous effort estimation solutions available, the estimates are frequently inaccurate. To increase the effort estimation accuracy, this thesis introduces a framework with novel methods and management practices for software project management professionals to estimate, collect and assess effort.

UNIVERSITY OF
EASTERN FINLAND

Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences

ISBN 978-952-61-0493-5