OULU 2014
A 634

ACTA UNIVERSITATIS OULUENSIS

Jarkko Hyysalo

SUPPORTING COLLABORATIVE DEVELOPMENT

COGNITIVE CHALLENGES AND SOLUTIONS OF DEVELOPING EMBEDDED SYSTEMS

UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU,
FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING,
DEPARTMENT OF INFORMATION PROCESSING SCIENCE

ACTA UNIVERSITATIS OULUENSIS
A Scientiae Rerum Naturalium 634

JARKKO HYYSALO

SUPPORTING COLLABORATIVE DEVELOPMENT

Cognitive challenges and solutions of developing embedded systems

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in the OP auditorium (L10), Linnanmaa, on 12 December 2014, at 12 noon

UNIVERSITY OF OULU, OULU 2014

Acta Univ. Oul. A 634, 2014

Supervised by
Professor Markku Oivo
Doctor Pasi Kuvaja

Reviewed by
Professor Cornelia Boldyreff
Professor Tomi Männistö

Opponent
Professor Rini van Solingen

ISBN 978-952-62-0601-1 (Paperback)
ISBN 978-952-62-0602-8 (PDF)

ISSN 0355-3191 (Printed)
ISSN 1796-220X (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2014

Hyysalo, Jarkko, Supporting collaborative development. Cognitive challenges and solutions of developing embedded systems

University of Oulu Graduate School; University of Oulu, Faculty of Information Technology and Electrical Engineering, Department of Information Processing Science

Acta Univ. Oul. A 634, 2014

University of Oulu, P.O. Box 8000, FI-90014 University of Oulu, Finland

Abstract

The development of embedded systems is becoming increasingly challenging; it is intellectually demanding knowledge work that requires collaboration among a wide range of skills. Software development is a largely cognitive activity, based on the worker's internal mental processes rather than on physical labour. Developers face several individual and team cognition-related challenges in their work, including complex decision-making and problem-solving processes. Therefore, it is suggested that the software development process should be modelled as a set of problem-solving activities.

This thesis proposes that supporting the cognitive work of collaborative development requires addressing the entire system's life cycle with practical solutions. In this work, the above-mentioned challenges are addressed in terms of communication and collaboration practices, knowledge management and coordination, and transparent tools and processes. Moreover, these solutions are integrated into a workflow that structures and supports the development process. Finally, a development process is outlined that addresses the decision-oriented nature of software development in such a manner that the necessary data is provided for decision points that guide and coordinate the development efforts.

A qualitative research approach has been chosen, and the work is based on interviewing industrial experts. Several cases were set up to define the state of the practice in industrial organisations developing embedded systems for different domains. Current challenges were identified and solutions were developed and validated in case companies.

The main result of the dissertation is a set of solutions integrated into the organisational workflow to support collaborative development. The main principles are that the necessary information must be provided and work and its objectives must be justified and put into the correct context. The industrial cases indicate that utilising the suggested solutions can improve collaboration among organisations and teams by helping disseminate and use the required information. Mitigating the cognitive burden speeds up the development work and reduces the effort required from developers and decision makers. In this manner, organisations may achieve better results, primarily because the produced data and results will fulfil their purposes better and provide less waste.

Keywords: cognitive support, collaboration support, collaborative development, embedded systems, software development

Hyysalo, Jarkko, Yhteistyön tukeminen ohjelmistotuotannossa. Kognitiivisia haasteita ja ratkaisuja sulautettujen ohjelmistojen tuotannossa
Oulun yliopiston tutkijakoulu; Oulun yliopisto, Tieto- ja sähkötekniikan tiedekunta, Tietojenkäsittelytieteiden laitos
Acta Univ. Oul. A 634, 2014
Oulun yliopisto, PL 8000, 90014 Oulun yliopisto

Tiivistelmä

Ohjelmistotuotanto nykymaailmassa muuttuu koko ajan haastavammaksi, kehitysprojektit ovat monimutkaisia ja hajautettuja sekä vaativat monialaista osaamista. Tiukat aikataulupaineet puolestaan tuovat mukaan oman problematiikkansa. Ohjelmistokehitys on suurelta osin kognitiivista työtä, jossa tarvitaan erilaisia taitoja ja eri alojen asiantuntijoita. Kognitiivinen työ tarkoittaa abstraktin tiedon käsittelyä ennemmin kuin fyysistä työtä. Ohjelmistojen kehittäjät törmäävät useisiin henkilökohtaiseen sekä ryhmätyöhön liittyviin haasteisiin, näistä esimerkkeinä monitahoinen tiedon käsittely, päätöksenteko ja ongelmanratkaisu. Onkin ehdotettu, että ohjelmistonkehitysprosessit ymmärrettäisiin ongelmanratkaisu- ja päätöspainotteisina prosesseina.

Tässä työssä ehdotetaan, että tukeakseen ohjelmistonkehitysprosessia koko tuotteen tekemisen elinkaari on otettava huomioon ja työntekijöiden roolit ja vastuut on linkitettävä kehitysprosesseihin sekä kehitysprosessin eri vaiheisiin. Havaittuihin kognitiivisiin ongelmiin ja tarpeisiin vastataan yhteistyö- ja kommunikaatiokäytännöin, tiedonhallinnan, läpinäkyvyyden, työnkulun, ja päätöspainotteisten prosessien kautta.

Tulokset kerättiin käyttäen menetelmänä laadullista tapaustutkimusta, ja työ perustuu useiden teollisten asiantuntijoiden haastatteluihin. Tutkimus toteutettiin useassa eri teollisuuden organisaatiossa. Aluksi määritettiin lähtötilanne organisaatioissa sekä kirjallisuuden perusteella, kartoitettiin ongelmat, jonka jälkeen kehitettiin tärkeimmiksi havaittuihin ongelmiin ratkaisuja.

Työn tuloksena esitetään joukko ratkaisuja, jotka yhdistetään organisaation työnkulkuun. Lisäksi esitellään päätöksentekoon painottuva kehitysprosessi, jonka lähtökohtana on havainto, että vaadittavien tehtävien sekä työn tulosten on vastattava oikeaan tarpeeseen – työlle ja halutuille työn tuloksille on annettava riittävät tiedot, perustelut, päämäärä sekä oikea konteksti. Tapaustutkimukset osoittavat, että työn tulokset parantavat organisaatioiden välistä yhteistyötä helpottamalla oikean tiedon keräämistä, saamista ja käyttöä. Lisäksi ylimääräisen kognitiivisen taakan vähentäminen nopeuttaa kehitystyötä ja keventää kehittäjien ja päätöksentekijöiden työkuormaa. Täten organisaatiot voivat saavuttaa parempia työn tuloksia lähinnä siksi, että tuotettu tieto ja tulokset vastaavat paremmin tarpeisiin.

Asiasanat: kognitiivinen tuki, ohjelmistotuotanto, sulautetut järjestelmät, yhteistyö, yhteistyön tukeminen

To my family

Acknowledgements

I would like to express my gratitude to my supervisors, Professor Markku Oivo and Dr Pasi Kuvaja. I am also grateful to Professor Mikko Siponen who got me started with work on my thesis. You all gave me very valuable comments relating to my doctoral dissertation as well as guidance on article writing.

It has been a pleasure to work with Pasi on all of his research projects; I am grateful for him for all the opportunities he has given me, thus making this thesis possible. I also appreciate all the support from my co-workers, especially Sanja Aaramaa, Markus Kelanti and Nebojša Taušan, a team that worked closely with me and provided valuable feedback and comments.

Moreover, I would like to thank the company representatives who participated in this research, especially Jari Lehto who provided an endless source of interesting industrial problems to be solved and with whom I had numerous discussions about topics that were relevant for my thesis.

I would also like to thank the pre-examiners of this dissertation, Professor Cornelia Boldyreff of the University of Greenwich and Professor Tomi Männistö of the University of Helsinki, for their valuable comments and recommendations.

Finally, I would like to thank my family, Tiina and Elsa, for their love and support.

Oulu, September 2014 Jarkko Hyysalo

List of abbreviations

CSCW	Computer Supported Cooperative Work
DC	Decision Criteria
DfX	Design For eXcellence
DP	Decision Point
ICT	Information and Communications Technology
R&D	Research and Development
RQ	Research Question
SME	Small and Medium Enterprise

List of original publications

This thesis is based on the following publications, referred to throughout the text by their Roman numerals:

I Hyysalo J, Parviainen P & Tihinen M (2006) Collaborative embedded systems development: Survey of state of the practice. In Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), March 27-30, 2006, Potsdam, Germany: 130–138.

II Liukkunen K, Lindberg K, Hyysalo J & Markkula J (2010) Supporting collaboration in the geographically distributed work with communication tools in the remote district SME's. In Proceedings of the 5th IEEE International Conference on Global Software Engineering (ICGSE), Princeton, NJ, USA, August 23-26, 2010: 155–164.

III Hyysalo J, Aaramaa S, Similä J, Saukkonen S, Belt P & Lehto J (2009) A new way to organize DFX in a large organization. In Proceedings of Profes 2009, 10th International Conference on Product Focused Software Development and Process Improvement, June 15-17, 2009, Oulu, Finland, Lecture Notes in Business Information Processing, Volume 32: 275–189.

IV Kelanti M, Hyysalo J, Välimäki A, Kuvaja P & Oivo M (2013) A case study of requirements management: Toward transparency in requirements management tools. In Proceedings of the 8th International Conference on Software Engineering Advances (ICSEA 2013), October 27-31, 2013, Venice, Italy: 597–604. ISSN: 2308-4235. ISBN: 978-1-61208-304-9.

V Hyysalo J, Lehto J, Aaramaa S & Kelanti M (2013). Supporting cognitive work in software development workflows. In Proceedings of Profes 2013, 14th International Conference on Product-Focused Software Process Improvement. June 12-14, 2013, Paphos, Cyprus: 20–34.

VI Hyysalo J, Kelanti M, Lehto J, Kuvaja P & Oivo M (2014) Software development as a decision-oriented process. In: Software Business. Towards Continuous Value Delivery: 132–147.

All of the articles have been published as full papers in conference proceedings, with a scientific peer review process. The author of this dissertation was the primary author of publications I, III, V, and VI. In those four articles, the researcher was responsible for formulating the research problems, developing the theoretical bases, formulating the research questions, coordinating the collection of empirical material, analysing the material, drawing conclusions, and being the primary author. The co-authors supported the work in the roles of reviewers and advisors on the structure, logic, and contents, as well as by supporting the data collection. The researcher contributed significantly to the writing of all sections of articles II and IV, as well as contributing to their research questions and empirical

work by participating in the design and implementation of the interviews and analysing the empirical results.

Table of contents

1 Introduction

In today's world, software systems are becoming ever more complex, and as a result, their development has become increasingly challenging. This concerns especially embedded systems, where software is part of a system that consists of both hardware and software. This dissertation studies collaborative development of embedded systems, identifies challenges, and proposes solutions to those challenges. The main focus is on addressing cognitive challenges and proposing solutions to them in order to help software developers in their daily work.

1.1 Background

The development of embedded systems is usually a multidisciplinary effort, and as noted by Helo (2004), it is often a highly complicated process with tight time-to-market requirements. As a consequence, successful embedded products are typically developed in collaboration among different stakeholders with different skills, such as engineers, industrial designers, and marketing personnel (Cooper *et al*. 2004, Gupta *et al*. 2007).

Software development involves challenging and intellectually demanding knowledge work (Robillard 1999, Bjørnson & Dingsøyr 2008). This knowledge work is largely a cognitive activity (Sung 2005) based on the worker's internal mental processes, rather than on physical labour. Noble (2004) stressed that cognitive perspectives are fundamental factors for successful collaboration. Therefore, knowledge work and cognitive aspects should be supported properly in order to provide good results.

Support for cognitive work is needed when working with abstract knowledge in order to meet the various challenges: for example, knowledge is not easily transferred, unless it is made explicit; knowledge elements are context-specific; and cooperation is needed due to the cognitive limitations of humans—one cannot know everything (van Leijen & Baets 2003). Cognitive skills are also required in order to respond to changes. Changes and unexpected events require creativity and human problem-solving skills to overcome and solve them (van Merriënboer 1997).

In this work, the cognitive activities and cognitive processes involved include making observations; reasoning; processing information; learning, understanding and remembering information content; using information systems; and conceptualising. Developers face several individual and team cognition-related

challenges in their work, such as complex decision making and problem solving, handling vast amounts of information, creating a shared understanding, and information and knowledge sharing.

It has even been proposed that the software development process should be modelled as a set of problem-solving activities (Wild *et al*. 1994). Problem solving, in turn, can be seen as decision making (Aurum & Wohlin 2003), as several stakeholders need to communicate and make numerous decisions during the development process.

One way to structure the processes that are built to ensure that activities in an organisation are performed consistently and reliably (Mangan & Sadiq 2002) is to develop workflows (WFMC 1999). Workflows can be described as a sequence of working steps or logically related tasks, including the use of resources to achieve a common goal—transforming a set of inputs into outputs of value to stakeholders. However, it is not easy to plan the work in detail beforehand (Buckingham Shum 1998), as many of the processes are complicated in nature, involving a large number of tasks, performers, and coordination constraints. Real-life work is more varied and more dynamic than current static workflow models can support. In addition, workflows do not handle exceptions well (see e.g. Jennings *et al*. 1996, Kwan & Balasubramanian 1997, Yu & Schmid 1999, Klein & Dellarocas 2000, van der Aalst & Basten 2002, Adams *et al*. 2006, Minor *et al*. 2011) and cognitive work and cooperation are not properly supported in current workflow models (Zhuge 2003, Wang & Wang 2006). Cognitive cooperation occurs when individuals learn from each other, make abstractions, solve problems, and use their experience and skills (Gaines 1977, Goel 1997, Zhuge *et al*. 1997, Zhuge 2003).

In sum, a practical solution is needed for supporting cognitive work in the collaborative development of embedded systems. For example, Hyysalo *et al*. (2006) proposed a development process that describes the roles and responsibilities of the different stakeholders, mapping them to each development phase and activity within the context of the organisation involved in the collaboration.

In this dissertation, comprised of six individual articles and this summary, ways to support the process will be defined in order to address the identified problems associated with collaborative work. In other words, the developed artefact of this work is a set of assets to support the collaborative development of embedded systems, especially from a cognitive point of view.

Requirements engineering is used as an example of one of the phases wherein the cognitive issues are easily recognised and support for cognition is especially important. Walia *et al*. (2006) reported that a significant proportion of requirements errors are caused by human cognition issues. Thus, two of the articles have a strong focus on the requirements engineering phase and promote cognitive support through the concept of Design for eXcellence (DfX) (Bralla 1996), as well as transparency and awareness in requirements management. DfX is an approach that can be utilised to understand the different stakeholders and their needs and to provide practices, guidelines, and previously tried solutions to development problems (Bralla 1996, Möttönen *et al*. 2009, Lehto *et al*. 2011). Transparency and awareness of the tools and processes available provide an understanding of the project and ensure that individual contributions are relevant to the overall objectives (Sarma 2005, Jiménez *et al*. 2009, Omoronyia *et al*. 2010).

The results of this dissertation will support developers' cognitive work, which is the result of interactions among individuals, artefacts, and resources in their environment (Hollan *et al*. 2000). To support cognitive work, the aim is to reduce the cognitive burden—the burden of keeping unnecessary factors in mind, such as how to use tools, how to link information between tools, and how to search for relevant data. Cognitive support is offered in several ways, as suggested by Kuutti (1995): automating routines, supporting communication, making things visible, making objects manipulable, supporting transformative and manipulative actions, supporting sense-making actions within an activity, and supporting learning.

1.2 Objectives and scope

The motive for this work arose from the realisation that previous studies notwithstanding, there are still problems associated with the collaborative development of embedded systems, especially in supporting cognitive work. There is a clear need for addressing cognitive activities and processes in software development. Thus, the research problem in this dissertation is:

> *How to provide cognitive support for the collaborative development of embedded systems?*

There are several possible ways to approach the research problem in detail. In this work, the research problem is addressed from six complementary viewpoints—

collaboration practices, communication, managing and coordinating knowledge, transparency and awareness, cognitive workflows, and a decision-oriented development process. The selected perspectives were formulated into subsequent research questions (RQ). The primary research question for each paper is presented in Table 1.

Table 1. Overview of research papers

Article	RQ#	Title	Primary research question
I	RQ1	Collaborative embedded systems development: Survey of state of the practice	What are the problems of and solutions to collaborative development?
II	RQ2	Supporting collaboration in the geographically distributed work with communication tools in the remote district SME's	How to support communication in distributed development?
III	RQ3	A new way to organize DfX in a large organization	How to manage and coordinate knowledge in embedded systems development?
IV	RQ4	A Case Study of Requirements Management: Toward Transparency in Requirements Management Tools	How does transparency support the collaborative work and creation of shared understanding?
V	RQ5	Supporting cognitive work in software development workflows	How is cognitive support provided in workflows?
VI	RQ6	Software development as a decision-oriented process	What type of process guides and controls the collaborative development?

These research questions are related to each other, even if they have different focuses. They all complement each other and contribute partially to defining cognitive support for collaborative work. The research questions are answered in detail in the conference articles of this dissertation and their contributions are outlined in this dissertation's summary. The positioning of the articles is presented in Fig. 1.

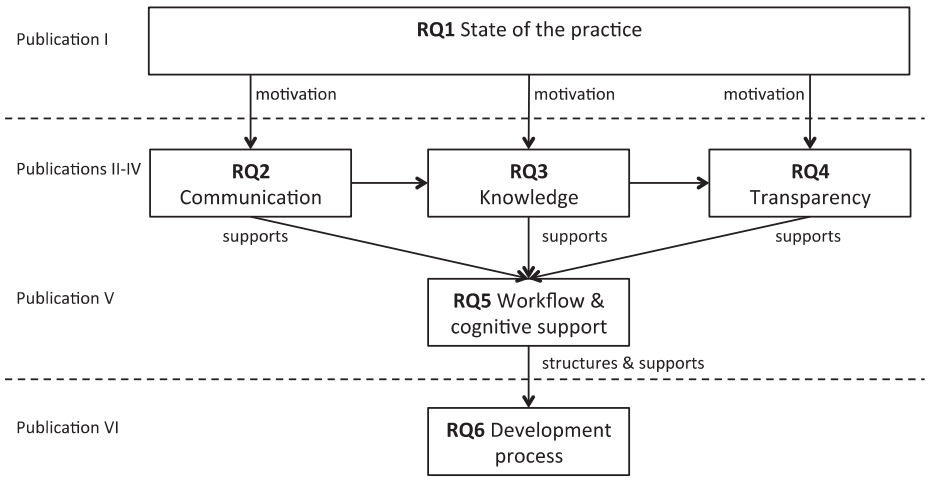

Fig. 1. Positioning of the articles

The aim of this dissertation is to provide cognitive support for developers' daily tasks while they work in collaborative settings. Solutions are presented to reconcile tools, practices, and ways of working in order to enhance the collaborative work. A workflow that structures the development process is developed to bring together all the aforementioned elements.

Article I provides the motivation and creates the basis for the study, while articles II through IV examine the different approaches to support the work of developers, with cognitive support being the main focus in this work. Article V integrates into the workflow the different support tools, methods, and practices discussed in the articles, which structure the development activities and processes discussed in article VI.

For practical reasons, the cases reported in articles III and IV focused mostly on the requirements engineering phase, to keep the research focus manageable for one researcher. Within the limits of a single doctoral dissertation, it is impossible to cover extensively the cognitive work of all software development activities. Requirements development and engineering were selected as case studies because they are representative of very knowledge-intensive work with multiple stakeholders, requiring plenty of support for cognitive work. In addition, requirements development and engineering were the most critical problem areas for the case companies.

1.3 Research approach

In this dissertation, an empirical approach (Basili 1993, Wohlin *et al*. 2012) with a qualitative case study method (Runeson & Höst 2009, Runeson *et al*. 2012) was chosen to gain a practical view of the topic. The motivation for selecting the approach was that the researcher had access to very experienced industrial experts and their views through qualitative interviews. The qualitative approach answers the questions of "why" and "how", which were regarded as critical and practical for both the researcher and the industrial case companies. In addition, action research intervention was conducted to iteratively develop and evaluate the results leading to Article V.

Semi-structured interviews, following the guidelines by Myers & Newman (2007), were used in all of the research papers. With the qualitative approach, the researcher was able to encourage the interviewees to express their viewpoints, opinions, and experiences freely, as much as possible and without limitations. The obtained material was analysed following the recommendations for thematic analysis in software engineering put forth by Cruzes & Dybå (2011a), and generalising conclusions were drawn based on the analysis.

Software engineering is a multidisciplinary subject that covers technical, language, and social issues. Software engineering is also human-intensive in nature—software cannot be manufactured. Software engineering requires human ingenuity and creativity. When studied, software engineering should be treated as a scientific discipline, which means through the application of scientific methods. One must understand and know the methods that are available and best suited for the phenomenon being researched. (Basili 1993, Wohlin *et al*. 2012).

When using an empirical research approach in software engineering, a model is proposed and evaluated through empirical studies. Empirical methods are traditionally tied to social sciences, as they are concerned with human behaviour. Depending on the purpose and conditions under which the empirical investigation is taking place, there are three main strategies: surveys, case studies, and experiments. (Basili 1993, Wohlin *et al*. 2012).

Survey is a method for collecting qualitative or quantitative information from or about people in order to explain, compare, and describe their attitude, behaviour, and knowledge. A survey gathers data using interviews and questionnaires targeting a phenomenon that occurred in the past. The data is analysed and the findings are generalised to fit the population from which the samples were selected. (Babbie 1990, Pfleeger 1994, Wohlin *et al*. 2012).

Case studies in software engineering are empirical studies that use multiple sources to investigate one (or a small number) instance of a phenomenon within a real-life context. Case studies focus on projects, activities, and tasks. Data is collected throughout the study by different means and from different sources. Action research is closely related to case study, but it is focused on and involved in the change process, while case study is purely observational. When the effects of a change are studied, action research can be classified as a case study method. Action research is used if the researcher actively participates in improvements; case study guidelines are applicable to the research part of action research. Action research is an iterative process in which theory-based diagnosis is followed by practical intervention through action planning, action taking, and evaluation. Learning from action and evaluation results in change, and the cycle is repeated until satisfactory results are obtained. Action research is one way theories are put into practice to help an organisation solve concrete problems, while at the same time expanding scientific knowledge. (Baskerville & Wood-Harper 1996, Runeson & Höst 2009, Runeson *et al*. 2012, Wohlin *et al*. 2012).

In software engineering, an experiment is an empirical study that manipulates one variable or factor of the studied setting, keeps other variables constant, and measures the effect of manipulation on the dependent variable. Experiments are most often conducted in laboratory conditions, guaranteeing a high level of control, while the experiment participants are selected from a population using a randomisation technique. (Wohlin *et al*. 2012).

While surveys and case studies can be qualitative or quantitative, experiments are quantitative by nature. Qualitative research gives the researcher a certain degree of freedom to plan and execute the research; however, the researcher is not completely free of his own values and limitations. Thus, full objectivity cannot be reached, as the researcher and the studied phenomena are connected. However, truly objective knowledge cannot exist, as all research is subjective in the sense that the researcher's understanding has an influence on the obtained results. (Tuomi & Sarajärvi 2006, Hirsjärvi *et al*. 2008, Eskola & Suoranta 2008, Wohlin *et al*. 2012).

The goal of qualitative research is to understand the studied phenomenon and to clarify its meaning and significance. Typically, data is collected directly from the field by interviewing and/or observing. Objects under study may often be limited in number; thus, the thoroughness of the study and quality of the input material are of utmost importance. The sample size must be high enough, and it must cover enough in relation to the intended analysis and interpretation.

(Hirsjärvi & Huttunen 1995, Eskola & Suoranta 1998, Marshall & Rossmann 1998, Patton 2002, Siggelkow 2007).

The aim of qualitative research is to understand a certain phenomenon, to describe how it functions, and to provide a theoretically solid interpretation of this phenomenon, with a goal of describing real-life realities (Denzin & Lincoln 2005, Hirsjärvi *et al*. 2008, Eskola & Suoranta 2008). The qualitative approach yields a rich picture of a respondent's point of view and in-depth knowledge about the subject matter (Denzin & Lincoln 2005).

Thematic analysis, a theoretically flexible approach for identifying, analysing, and reporting themes in qualitative research, is one of the most commonly used synthesis methods in software engineering. In thematic analysis, the data is organised and described in rich detail and interprets various aspects of the research topic. It can be used within various theoretical frameworks to report on the experiences, meanings, and realities of the participants. (Braun & Clarke 2006, Cruzes & Dybå 2011a, Cruzes & Dybå 2011b).

1.4 Research realisation and dissertation structure

As mentioned previously, an empirical and qualitative research approach was chosen for this work, as it enables data collection directly from the field. The empirical material for this dissertation was obtained through interviews, questionnaires, and observations. In addition, relevant company materials were used when provided. Surveys and case studies were selected as the main research methods. Both interviews and questionnaires were used as data collection methods in article I. Direct (e.g. interviews) and independent (e.g. document analysis) collection methods were used in the case studies reported in articles II–VI. In addition, in a study reported in article V, a prototype was developed and its use was observed through an action research intervention.

Interviews with over one hundred industrial experts make up the main basis of the original publications. The interviews were complemented with literature studies and workshops. Observations of the prototype piloting were used in one case, reported in article V.

The researcher participated in the materials collection, and more importantly, was responsible for conducting the analysis and drawing conclusions.

Table 2 presents the number of interviews, questionnaires and observations, and the number of companies studied for each article.

Table 2. Number of interviews, questionnaires and observations in each article

Article	Title	Key data collection methods	Number of companies
I	Collaborative embedded systems development: Survey of state of the practice	12 interviews and 7 questionnaires	6
II	Supporting collaboration in the geographically distributed work with communication tools in the remote district SME's	30 interviews	13
III	A new way to organize DfX in a large organization	20 interviews	1
IV	A Case Study of Requirements Management: Toward Transparency in Requirements Management Tools	11 interviews	1
V	Supporting cognitive work in software development workflows	35 interviews and 36 observations	1
VI	Software development as a decision-oriented process	46 interviews	2

The research was conducted in four phases, all of which involved industry practitioners and researchers from different organisations. The case companies operated in several industrial areas, such as automation, consumer electronics, and telecommunications, representing divergent software systems business areas. All of the companies produce embedded systems and work in a distributed development mode. Small, medium, and large companies were included. The interviewees represented different levels in the case companies, from "floor-level" to upper middle management, such as senior managers, project managers, software developers, and testers. Most had several years of experience. Each study phase resulted in one or more conference articles. As several different case companies and domains are represented in the study, it enables drawing conclusions over different domains and contexts.

An overview of the research strategy and proceeding is presented in Fig. 2.

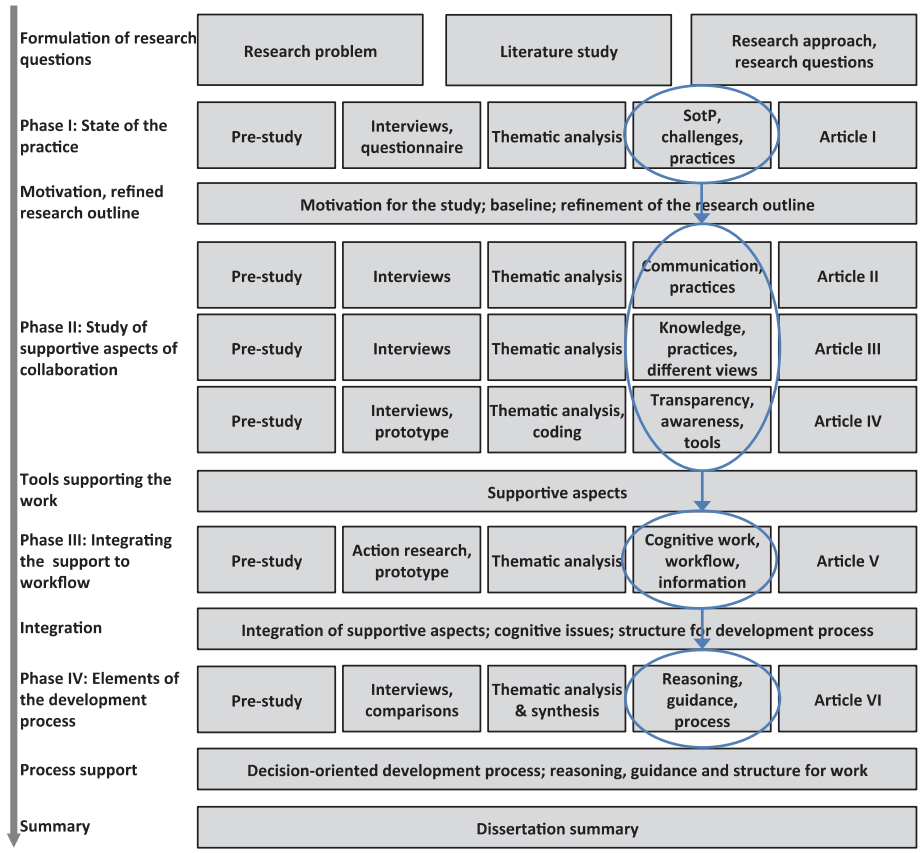

Fig. 2. Proceeding of the research according to study phases

Phase 1 – State of the practice

The original research problem was the starting point. During the first phase, the literature was studied in order to understand the key concepts and existing theories. Based on this work, further research questions were created and a research approach was selected. In addition to conducting interviews, a survey was created and sent to the companies. After the completion of the first phase, a motivation for further studies emerged, and further research questions were identified. The first phase resulted in article I, which presented the state of the practice of collaborative work and provided the baseline for follow-up studies.

The research process is presented in Fig 3. A similar process was used for all studies in phases I, II, and IV, producing articles I–IV and VI.

Fig. 3. Research process of articles I–IV and VI

The chosen topic was first explored through an extensive literature study. When the case companies were able to provide materials (such as process descriptions, product descriptions, best practices, templates, and tool-related materials), they were used as well in order to better understand the case company, its development environment, and its challenges. Workshops were held during this phase to guarantee a shared understanding between the researchers and case companies regarding the study objectives. Based on the understanding obtained, the research was focused on the most critical issues. The interview structure and questionnaires were then formulated, potential interview candidates were selected with the help of company representatives, and the interviews were conducted. Different companies were included in the study in order to obtain as broad a coverage of the subject matter as possible. All of the interviews followed a semi-structured approach (Myers & Newman 2007). The interviews were conducted informally, in a qualitative manner, allowing the interviewees to explain their views and clarify the cases and topics as entities. The interviews were recorded and transcribed in order to ensure full utilisation of the views and opinions of the industrial experts. The interviews were then thoroughly analysed and compared to the literature, when appropriate. The thematic analysis steps suggested by Cruzes & Dybå (2011a) were followed. Finally, conclusions were drawn based on the analysis.

Phase 2 – Study of supportive aspects of collaboration

The second phase consisted of three studies focusing on different supportive elements of collaborative and cognitive work. The selection of these elements was based on the work of the first phase and its results. These elements were studied in individual cases in different organisations. As a result of the second phase, three articles were produced that highlighted the importance of the selected

elements, as well as proposed solutions that can be used to support cognition in a collaborative development.

Phase 3 – Integrating the support to workflow

The third phase integrated the previously defined cognition-supporting elements into a workflow, addressing the cognitive issues hindering the work of software developers. In addition, the defined workflow provides structure and information flows for the collaborative development process. In this phase, a prototype was developed and validated in an action research intervention within a case company in order to test the developed supportive elements in a real environment.

The third phase, which produced article V, followed an action research cycle. The research started with a pre-study, wherein the topic was familiarised first, and then the initial construct was developed. The case company was studied to understand the state of the practice, identifying the current challenges. After the pre-study, the action research cycle commenced. Action research was applied in order to see the construct in practice, to improve it, and to evaluate the outcome. Evaluations were conducted by implementing the construct and applying it in its intended settings, thus providing empirical evidence of its use. The use of the prototype was observed by one researcher who made observational notes; in addition, the sessions were recorded. The researcher also provided guidance when needed. The software developers and experts were encouraged to think aloud as much as possible, and after each evaluation phase, experiences were discussed in the monthly team meeting. Gathered feedback, observed challenges, and identified bugs in the prototype were recorded. The researchers, prototype developers, and a case company representative held weekly meetings to review the results and decide on further actions. These actions provided rich data for analysis and for specifying learning. A workshop was then held at the case company to report the initial findings and gather feedback, which was incorporated into the final findings. More thorough description of action research with concrete actions taken and learning specified are presented in article V.

The action research cycle, shown in Fig. 4, was repeated three times, until the results were satisfactory.

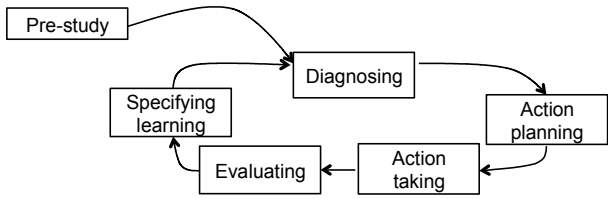

Fig. 4. Research process of article V (adapted from Susman & Evered 1978)

Phase 4 – Elements of the development process

The fourth phase examined, once again, the data from the previous studies and added new insights by cross-analysing the results of earlier studies and by analysing the results from a new viewpoint. Two companies were studied, representing two different domains: information and communications technology (ICT) and the automation industry. Thematic analysis was used on the combined data from the earlier studies. The data and results of the different cases were analysed and comparisons were made until the final set of topics was defined. The understanding obtained was then applied in defining a decision-oriented development process and its elements, and it was finally reported in article VI.

In each article, the research processes and conclusions of the results are discussed in more detail. All of the articles are qualitative in nature, utilising the principles of empiricism and applying a mainly thematic analysis. In these articles, qualitative questionnaires were used for interviewing experienced developers and managers. The obtained material was analysed and general conclusions were drawn based on the analysis. In addition, action research was applied for model development and evaluation, with the aim of solving a specific problem through an innovation. Finally, this dissertation thesis summarises the work, presents the contributions, and discusses the validity of the research as a whole and its results.

This dissertation consists of six individual articles and this summary. The rest of the summary is organised as follows: Chapter 2 studies the related works; Chapter 3 introduces the six articles and discusses their main results; Chapter 4 discusses the research contributions of the individual articles, their study limitations, and future work; finally, Chapter 5 summarises the research.

2 Related work

Several cognitive challenges in collaborative software development can be addressed with improved collaboration and communication practices, implicit and explicit knowledge management, better awareness tools and processes, cognitive workflows, and processes supporting coordination and negotiation (see e.g. Kuutti 1995, Zhuge 2003, Noble 2004, Wang & Wang 2006, Espinosa *et al*. 2007, Jalote-Parmar *et al*. 2010, Treude & Storey 2012). These concepts are used in this work to address cognitive support, as shown in Fig. 5.

Many other concepts can also be seen as relating to cognitive support for collaboration, such as distributed work, global software development, decision support systems, virtual collaboration tools, learning, and computer-supported cooperative work (CSCW). This dissertation does not address all of those areas individually, but focuses on viewing the support for collaborative software development from a cognitive point of view. However, CSCW deserves a short mention, as it has much in common with this work and takes a multidisciplinary approach to the study of collaborative work and work practices. This field includes various aspects, such as the ethnographic, social, technological, and theoretical issues that enable group work (Ahmed & Tripathi 2010). CSCW is more user-oriented, while workflows are more process-oriented; however, workflow systems are often also CSCW systems (Ahmed & Tripathi 2010).

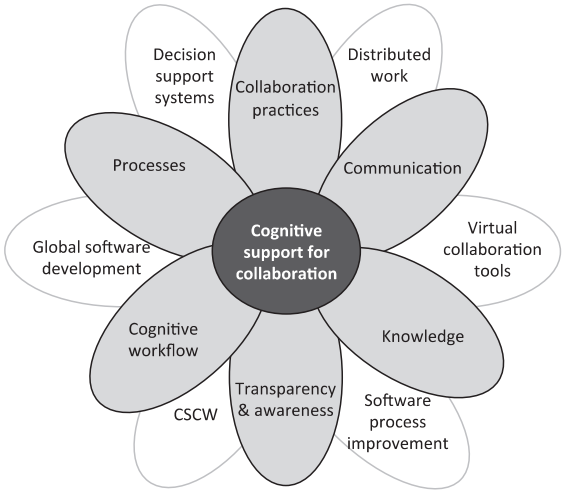

Fig. 5. Position of the dissertation in relation to theory

Figure 5 positions this research (grey areas) in the field of related topics (white areas), showing some examples of the research areas that could be mentioned or applied in this work. Many ways to integrate the approaches exist, and one of those is presented in this work. In this work, selected approaches are used together to build cognitive support for collaborative software development. The aim of this work is to describe how work can be organised and how it can be supported through processes and workflows, and how decision points and related decision criteria guide, structure, and coordinate the development efforts. As cognitive support can be studied from several different viewpoints, it is impossible to cover cognitive work extensively from all those viewpoints within the limits of a single doctoral dissertation. Thus, some relevant areas have been left out for practical reasons, though they might have been considered had the scale of this work been wider.

2.1 Collaborative development

The current trend is that the development of embedded systems is highly complicated, with tight time-to-market requirements (Helo 2004). Consequently, the development is conducted in collaboration among various stakeholders (Gupta *et al*. 2007, Omoronyia *et al*. 2010, Lanubile *et al*. 2010), such as subcontractors, third-party suppliers, and in-house developers. Collaborative development is defined as activities involving two or more organisations, departments, or customers, combining their competencies and technologies to create new shared value and managing their respective costs and risks (Welborn & Kasten 2003).

Collaborative development offers several advantages, such as potential savings in development times and costs and being close to customers (Carmel & Agarwal 2001, Herbsleb *et al*. 2005). However, it comes with a cost—collaborative development is highly challenging, often emphasising the challenges met in single-site development and adding new ones; for example, dispersion of the development teams, which alone places high demands on communication, teamwork, and work methods (Olson *et al*. 1998, Herbsleb *et al*. 2001, Herbsleb & Moitra 2001, Damian & Zowghi 2003, Herbsleb & Mockus 2003, Paasivaara & Lassenius 2004, Noll *et al*. 2010). In addition, different time zones and distances increase communication difficulties (Damian & Zowghi 2003); different sites might use different tools, processes, and practices; and work habits and organisational cultures are often different as well (Herbsleb 2007).

Thus, the development process differs significantly from the single-site development process, and it is more challenging.

Coordination and communication become difficult in collaborative development as well, and the creation of awareness suffers, diminishing productivity and product quality. To tackle these challenges, proper tools, processes, and methods that address the entire product lifecycle are required. (Damian & Lanubile 2004, Herbsleb 2007, Jiménez *et al*. 2009).

Developing and maintaining a shared understanding is also necessary, and achieving a shared understanding requires considerably more effort in distributed development than in co-located settings (Omoronyia *et al*. 2010). Furthermore, cognitive perspectives become fundamental factors for successful collaboration by supporting multiple aspects of collaboration (Noble 2004). Several tools are available to address the various collaboration issues, and the literature offers best practices (see e.g. Whitehead *et al*. 2010). Those supportive assets should be available for developers to use to manage their daily tasks, and they should be integrated into the development environment in order to guarantee easy and seamless use.

2.2 Communication

Software development requires a vast amount of communication (Jiménez *et al*. 2009), and the distributed nature of development usually means that either the experts must travel to other sites to share their knowledge or meetings have to be arranged in virtual space. Virtual presence, however require good communication connections and tools to support collaboration.

Communication is a mediating factor in coordinating and controlling the collaborative work (Carmel & Agarwal 2001). In fact, communication is often deemed the most critical factor in collaboration and must be arranged properly in order to have successful collaborative development. In particular, requirements engineering is one of the most challenging processes in distributed development, as it requires vast amounts of multidisciplinary communication and interaction (Cheng & Atlee 2007, Monasor *et al*. 2010).

While communication infrastructures are available and current communication technology offers tools that enable communication among geographically distributed teams, obstacles still exist, such as numerous management challenges due to limited interaction opportunities (Grinter *et al*. 1999, Herbsleb & Moitra 2001, Pauleen 2003), language barriers (Pauleen &

Yoong 2001, Sarker & Sahay 2003), and time zone differences (Carmel & Agarwal 2001).

Knowledge is central to collaboration, and it must be distributed among the development teams (Noble 2004); hence, there is a need for communication. It can be concluded that there is no communication without cognition, as communication between "information processing units" includes several concepts of cognition. Communication includes concepts such as comprehension, reasoning, information processing, learning, and remembering. Furthermore, knowledge is a result of cognitive processing (Alavi & Leidner 2001).

The communication challenges of geographically dispersed enterprises can be looked at through existing theories, such as the media richness theory (Daft & Lengel 1986), media synchronicity theory (Dennis & Valacich 1999), and social presence theory (Short *et al*. 1976). These theories propose that rich information, shared understanding, and awareness information are needed for complex knowledge tasks. However, a gap has been identified between software systems experts and users of software systems, which, according to Adams *et al*. (2005), results from the realisation that technologies are designed and implemented on the basis of assumptions without enough consideration of the users.

One of the problems is that developers have to rely on various tools and formats that do not necessarily follow any communication standards or may not provide all the necessary cognitive support, which can lead to misunderstandings. When combined with a complex infrastructure, this has been reported to decrease both the frequency and quality of communication, and ultimately, productivity. To mitigate these issues, tools, processes, and methodologies are required. (Jiménez *et al.* 2009).

Thus, there is still a need for new ways (tools, processes, methods, etc.) to support collaboration and to communicate and disseminate information among developers, taking into account the real requirements of the users, developers, work, and processes.

2.3 Design for excellence in managing and coordinating knowledge, practices, and views

The embedded systems sector is currently facing several challenges. Software development companies seek more effective and efficient processes to address such challenges as yielding high customer satisfaction, changing needs and requirements, short development times, and tight schedules (e.g. Weber &

Weisbrod 2003, Jiao & Chen 2006, Birk & Heller 2007). Fulfilling customers' needs is necessary; however, the needs might be very customer-specific, possibly requiring a lot of customisation. While standardised processes and products contribute greatly to efficiency and quality, customisation and flexibility are also required. This raises the question of how to manage changing requirements and customer needs while taking into account the constraints of the design process and still bring a desirable, quality product to market—how do we find a balance between standardisation and customisation? Another issue is that of determining how to design and develop products while recognising and valuing the needs of various stakeholders adequately, especially considering the needs of internal and external stakeholders equally.

Efficient requirements engineering and design is a demanding task. DfX offers a way to bring together different views and harmonise practices. It is important to adequately address the needs of both internal and external stakeholders. External customers often bring in revenue directly and are more visible, and they are often valued over internal customers (Lee & Billington 1992). Yet, internal stakeholders have a huge impact on the effectiveness and efficiency of the product creation and delivery process. The literature has also recognised that addressing the needs of internal customers is a key element for successful product development (Cooper *et al*. 2004, Gupta *et al*. 2007).

DfX, which has been utilised in the industry for several years (Möttönen *et al*. 2009), is an approach that brings together different views and harmonises practices. DfX is a knowledge-based approach whose aim is to design products in a manner that maximises all of the desirable characteristics while at the same time minimising lifetime costs, including manufacturing costs (Bralla 1996). Some examples of desirable characteristics are quality, environmental friendliness, manufacturability, assembly, and testability reliability. To achieve these objectives, the product design process itself has to be excellent.

DfX can be seen as a means of improving product design and development processes, and eventually, the final product (Bralla 1996). DfX can also be utilised as a communication tool to disseminate information, collect best practices, and realise the implementation of best practices (Möttönen *et al*. 2009, Lehto *et al*. 2011). Utilisation of this type of knowledge can provide considerable support. For example, Henninger (1997) suggested that problem solving consists of the utilisation of past experience in an analogical situation. Alavi & Leidner (2001) recognised that due to the distributed nature of organisational cognition, it is important to transfer knowledge to locations where it is needed and can be

utilised. When DfX is used to collect and disseminate developers' experiences, it can be utilised as a knowledge base to support developers' tasks.

As knowledge is a result of cognitive processing, where information is converted into knowledge, one significant implication is that the creation of a shared understanding requires that individuals must share a certain knowledge base. Having shared knowledge also facilitates coordination and developer interactions. Shared knowledge about tool processes, products, domains, and team members is developed over time, and the utilisation of shared knowledge is furthered during development tasks. Shared knowledge helps developers understand different perspectives and creates common ground. (Alavi & Leidner 2001, Herbsleb 2007, Espinosa *et al*. 2007).

DfX can also be seen as a tangible way to coordinate and manage different views and to achieve functional integration. It also helps to address the desired goals of the organisation: effective cost management, delivery, service, environment, testing, and quality, to name a few. The main benefit can be understood as the ability to put different stakeholders on equal ground, addressing their views on even terms. (Möttönen *et al*. 2009, Lehto *et al*. 2011).

The need for practical solutions arises from the difficulty of addressing the different stakeholders equally and knowing how to provide the necessary requirements and share the obtained knowledge, guidance, and practices with the development teams.

2.4 Transparency and awareness of processes and tools

The importance of transparency of processes and tools is recognised in the literature (e.g. Beaudouin-Lafon & Karsenty 1992, Grinter 1995, Gutwin *et al*. 1996, Herbsleb 2007, Berggren & Bernshteyn 2007). Transparency and awareness that enables transparency have a great impact on the efficiency of product development and the quality of developed artefacts. For example, Jalote-Parmar *et al*. (2010) suggested that situation awareness—comprised of the three main elements of cognitive processes: perception, comprehension, and projection of action plan—is necessary for decision making. Noble (2004) also suggested that awareness is one of the key knowledge enablers.

Transparency furnishes the tools to provide, for example, information about the on-going status of the development process, awareness of the actions of others, easy access to relevant information, and a more visible process. Thus, also communication, decision making, and information utilisation are facilitated by

supporting human cognition. On the other hand, if developers have no knowledge of what others are doing, misunderstandings regarding communication content and motivation can easily result (Hyysalo *et al*. 2006). A startling realisation was that regardless of its importance, transparency is not always taken into account in requirements for tools supporting development. For example, the transparency aspect is seriously lacking in requirements management tools, and requirements engineering, if anything, necessitates a great deal of communication and misunderstandings can be costly.

Awareness is an important concept in transparency, and it has been suggested that awareness is the key to transparency (Beaudouin-Lafon & Karsenty 1992). Awareness can be defined as an understanding of others' activities, which also provides the context for one's own activities (Dourish & Bellotti 1992). Addressing the awareness requirements (see e.g. Damian *et al*. 2003, Storey *et al*. 2005, Espinosa *et al*. 2007) can have a positive effect, for example, on the following:

- Openness of communication and information sharing
- Visibility of and access to data, documents, expertise and resources, and work items
- Visibility of decision making and decisions
- Visibility of processes and tasks
- Transparency of collaboration
- Transparency of tools
- Coordination

Awareness is critical to collaboration in software development, as it enables the creation and maintenance of a shared, realistic understanding of the project, and it ensures that individual contributions are relevant to the overall objectives (Sarma 2005, Jiménez *et al*. 2009, Omoronyia *et al*. 2010). Developers needs all the necessary information, such as the history and the status of the projects, who is working with what, the roles and expertise of their colleagues, the interactions and dependencies in the work system, and the work items (Jiménez *et al*. 2009, Omoronyia *et al*. 2010). Furthermore, awareness information should be obtained in a passive, unobtrusive manner instead of requiring developers to maintain awareness actively (Sarma 2005).

Both processes and tools must support transparency and awareness (Omoronyia *et al*. 2010). However, before that can be achieved, it is necessary to identify the transparency and awareness requirements for the tools and processes.

2.5 Cognitive workflow

The development of software systems is increasingly challenging and intellectually demanding knowledge work (Robillard 1999, Bjørnson & Dingsøyr 2008) that requires a wide range of skills. Software development is also information-intensive knowledge work (Nakakoji *et al*. 2010), which is a largely cognitive activity based on the developer's internal mental processes, rather than physical labour. Understanding software engineering as a developer-centred creative knowledge task puts the focus on cognitive and social processes (Nakakoji *et al*. 2010).

Sung (2005) stated that software development is a cognitive process, and Nakokoji *et al*. (2010) suggested that "*software development is about information, generating information, and making information artefacts*". In order to carry out knowledge-intensive tasks and solve problems, developers must understand both the current state and the goal state, and have a way to reach the goal. This understanding provides the basis for problem solving and task implementation. In addition, team cognition and knowledge (including long-term knowledge and awareness) support coordination and improve developer interactions, as developers can understand and anticipate what others do (Espinosa *et al*. 2007).

However, it is not a trivial task to have and understand all this information. In real life, work has many variables, changes and unexpected events occur, vast amounts of data must be handled, and innovative solutions are needed (Kwan & Balasubramanian 1997, Klein & Dellarocas 2000, Mangan & Sadiq 2002). There are substantial demands on developers' cognitive capabilities (van Merriënboer 1997), and reducing that cognitive burden—the burden of keeping unnecessary things in their minds—is important.

When a developer-centred approach to software engineering is taken, the purpose of collaborative software development environments should be to facilitate and nurture developers' creative knowledge processes (Nakakoji *et al*. 2010). Companies implement, for example, workflows to help their developers manage processes, transfer the work and data from one to another, and help establish a logical order for task implementation. However, traditional workflow approaches are static and do not address the changes and unexpected events that inevitably occur in demanding knowledge work; moreover, workflow models lack cognitive support (Jennings *et al*. 1996, Klein & Dellarocas 2000, van der Aalst & Basten 2002, Zhuge 2003, Minor *et al*. 2011).

Information visualisation and situation awareness are needed in workflows (Jalote-Parmar *et al*. 2010). For example, Grambow *et al*. (2011) discussed lack of situation awareness and the importance of connecting the abstract high-level processes to developers' concrete actions and workflows. Suggested solutions include providing an awareness of context (Omoronyia *et al*. 2010) and of others' actions, which makes it possible for developers to structure their interactions and cooperative processes and to provide a context for one's own activities (Dourish & Bellotti 1992, Robertson 1997). The context of tasks provides a birds-eye view of the work, wherein tasks are put into their place in the whole context, and visualises the work and interdependencies of activities (Kulkarni *et al*. 2012).

It is crucial to understand the information needs of developers and to address those needs. Providing the relevant information for developers helps to reduce the cognitive burden, and tools, processes, and practices should support the information provision.

2.6 Decision-oriented software development

Software development is complex effort that can be modelled as a set of problem-solving activities (Wild *et al*. 1994), while problem solving, in turn, is in essence decision making (Aurum & Wohlin 2003). Thus, we can understand software development as a decision-oriented process.

Processes are built to guarantee that activities are performed consistently and reliably (Mangan & Sadiq 2002), the four primary components are objectives, tasks, performers, and constraints (Sadiq & Orlowska 1999). However, design problems often result from ill-defined goals and evaluation criteria (Guindon 1990). Furthermore, defining the process strictly beforehand may be impractical. Interaction with the environment, the activities, and the underlying business logic sets the order, instead of a predetermined, static process schema (Wang & Wang 2006).

The constant changes and unexpected events that are inevitably encountered in today's turbulent world lead to the need for a development process that has the ability to adapt to different situations. The changes and unexpected events also require creativity and human problem-solving skills to overcome and solve them (van Merriënboer 1997). An opportunistic design behaviour is proposed (Guindon 1990), where processes are modelled at a high level and knowledge-intensive tasks are embedded as black boxes, without too much detail (Abecker *et al*. 2002).

In trivial cases, simple problems can be solved with a top-down approach, following the predefined process. However, non-trivial problems often cause or require deviations from predefined top-down approaches. This has been recognised as an inherent and important aspect of solving non-trivial design problems. In practice, this means that the development process should not be too strictly defined and that there should be room for innovation and problem solving. (Guindon 1990, Buckingham Shum 1998).

Development is typically carried out in collaboration among several stakeholders (Cooper *et al*. 2004, Gupta *et al*. 2007, Pahl *et al*. 2007, Zeidler *et al*. 2008, Jiménez *et al*. 2009, Treude & Storey 2012), where each provides his own contribution towards the common goal. These contributions must be integrated into a single product, and parts must interoperate properly; furthermore, production must be synchronised, as there are many dependencies between tasks and persons (Espinosa *et al*. 2007). Therefore, it is paramount that all partners understand exactly what is expected of them, and the efforts must be coordinated and synchronised. When accurate decision criteria are provided and the results are checked against them regularly, the developers can do their work to fulfil their part, and various contributions—accomplished concurrently—can be synchronised and coordinated.

Recognising the importance of decision making as a way to guide development is a way to address the aforementioned challenges and to provide synchronisation and coordination to the development process. A decision-oriented approach can provide the decision criteria and degrees of freedom required for innovative problem solving.

3 Research contribution

This chapter briefly summarises the main results for and contribution of each article. The articles present perspectives that support collaborative development and reduce the cognitive burden of software developers.

3.1 Article I: Collaborative embedded systems development: Survey of state-of-the-practice

The first article addresses research question 1: *What are the problems of and solutions to collaborative development?* The article is based on twelve interviews and seven questionnaires, administered in six companies. The companies represent several divergent embedded software business areas, such as telecommunications, IT services, and consumer electronics. The interviews and questionnaires were qualitative, and the results were analysed using thematic analysis.

Article I describes the state of the practice in collaborative embedded systems development from the perspective of multiple organisations, and it reviews current collaborative practices, identifies challenges in collaborative work, and proposes solutions to those challenges. The aim of this article was to support effective collaboration among organisations by proposing solutions to support distributed development. The solutions were mapped to identified challenges in order to gain a view of current collaborative practices, in order to determine the most problematic or critical issues related to collaborative work, as well as the most important areas that should be the focus of research activities.

The findings of the industrial survey and the experiences gathered from the literature provided an insight into the state of the practice in collaborative embedded systems development. The study presents problems of and solutions for collaborative software development, and reasons for collaboration and different collaboration modes are discussed.

The study revealed that the most common collaboration mode within large companies developing embedded systems was subcontracting. Often, the organisation was split based on product structure, defined by product requirements or architecture. Body-shopping and distributed development was also used frequently. The following collaboration activities were identified in the study:

- Joint development agreements, especially among larger organisations
- Cooperation with domain experts
- Collaboration of one company with two others wherein the companies support each other in specialised expertise areas
- Participation in domain-specific forums, e.g. influencing standards
- Outsourcing maintenance
- Subcontracting hardware development to third parties

The collaboration mode was selected on a case-by-case basis, without clear rules or guidelines for any of the companies. The main reasons for collaboration were:

1. To reduce development costs
2. To acquire competence (technology competence or knowledge of a specific market)
3. To avoid investing in the company's non-core competence areas

The general risks, regardless of the mode of collaboration, were identified in the study as openness of communication among partners, unclear assignments, trust among partners, agreeing on intellectual property rights, and the reliability of the partners' development schedule. Further considerations arose from the continuity of collaboration, the quality of the acquired product, and competence issues.

The study identified fluency of co-operation, good understanding among partners of each other's work, mutual benefit from collaboration, and partners complementing each other's expertise as positive factors.

In most cases, no specific collaborative development tools were used. However, the same tools were often used by partners for specific activities, especially when specialised tools were required. It is noteworthy that most tools did not to support collaborative development. In particular, there was a need for better change management tools, and although no specific collaboration tools were used much, there was an apparent need for awareness- supporting tools.

The most critical areas in the collaboration were contracting, change management, requirements development, and requirements management. On the other hand, the areas most commonly seen as non-critical were software implementation, improvement processes, and human resource management.

The study revealed that the approaches to collaborative work represented by the literature and industrial practitioners were different. While the industry's focus was on technical aspects and detailed problems regarding engineering practices,

the literature focused on solutions to more general issues, such as communication and team building.

The survey identified several solutions, especially for management and support practices. On the other hand, the literature analysis found only a few solutions to engineering practices. The study concluded that practical solutions should be provided wherein the development process is defined by describing the roles and responsibilities of the different parties and mapping them to each development phase and/or activity within the context of organisations developing software in collaboration.

3.2 Article II: Supporting collaboration in the geographically distributed work with communication tools in the remote district SME's

The second article addressed research question 2: *How to support communication in distributed development?* Article II also complemented the large organisations view in research question 1 with the small and medium enterprise (SME) perspective, while the main focus was on discussing the communication aspects more thoroughly in order to address research question 2. The article is based on thirty qualitative interviews in thirteen companies operating in the domains of software and information systems development, web systems, consulting, and education. The data was analysed using thematic analysis.

Interviews conducted in geographically distributed software development companies were analysed against the literature. The analysis showed that the results are in line with theories of media richness, media synchronicity, and social presence, which propose that the more uncertain and ambiguous a task, the richer the media supporting the task implementation should be.

Issues related to management challenges, technology gap, and communication in particular were identified. The lack of constant communication stressed the need for good planning and agreed-upon practices. Smaller companies seemed to prefer agile ways of working and ad-hoc communication. Coordination was also easier in small companies due to flexible management styles.

The study points out that solutions and practices developed for or used in large companies do not fit straightforwardly into smaller companies. Smaller organisations must adjust their activities to match their own purposes, principles, and goals. The study shows that communication practices in smaller companies

differ from those used in large companies. Mainly due to a lack of resources, tools must be easy to use, as there are no resources for hiring support staff or arranging training. In fact, the personnel did not recognise the value of this type of training. These findings lead to the conclusion that tools, practices, and processes must be easily understandable, consistent, and follow the agreed-upon standards and guidelines. They must match the users' expectations and the workflows. Recognition was recommended over recall, meaning that the systems must be transparent and that users should immediately understand what to do instead of needing to remember things, thus reducing the cognitive burden.

The study presented in article II shows the importance of agreed-upon practices, understanding each other, and transparency and awareness in distributed collaborative work. The results of this article form a basis for such activities as tool development, presented in articles IV and V. The results are also recognised as a good guideline for the development of processes and practices, as those also must be transparent, easy to understand, and fit the working processes.

3.3 Article III: A new way to organise DfX in a large organisation

In order to address research question 3—*How to manage and coordinate knowledge in embedded systems development?*—the article is based on twenty qualitative interviews in a single company from the ICT domain. In addition to the interviews, the case company also provided archival material, process and product descriptions, and other documentation. The results are based on a thematic analysis of the gathered material.

Article III studied the importance of addressing the stakeholders and their needs equally and presented the DfX concept as a way to weigh the needs of different stakeholders equally. Article III found that more attention is needed to appreciate the internal customers.

Traditionally, DfX is a part of a company's research and development (R&D) organisation and thus, managed by designers. However, it can also be distributed to other parts of the organisation, as shown in the industrial case in the article: DfX (criteria and guidelines) were managed from operations instead of R&D, while DfX managers represented various stakeholders. Management responsibilities should reside where they matter most, such as with the organisational unit that pays the costs of the development, thus making the concept and its improvement more visible and widespread throughout the

organisation. This offers some remarkable benefits compared to the traditional method, including:

- The development needs of the various organisational units and internal and external stakeholders are treated equally.
- The design criteria and guidelines of all relevant stakeholders are more visible and are taken into serious consideration.
- Requirements that concern multiple programs, product lines, or families are more visible to all stakeholders.

The study also applied a new point of view to development and the concepts of problem domain and solution domain. Traditionally, problem domain is the environment in which a problem is defined—usually a problem that is to be "solved" by the product and the by-products related to it—while solution domain is the area in which the solution to the problem is defined. Thus, the solution domain provides solutions to solve the challenges of the problem domain (Jacobson *et al*. 1999). Fig. 6 presents the traditional view of problem domain and solution domain with the two main groups of stakeholders, customers, and developers.

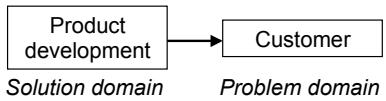

Fig. 6. Traditional relationship of domains and stakeholders

From the developer's point of view, the traditional setting can be turned upside down, and the situation examined as shown in Fig. 7, with DfX as a solution domain issue.

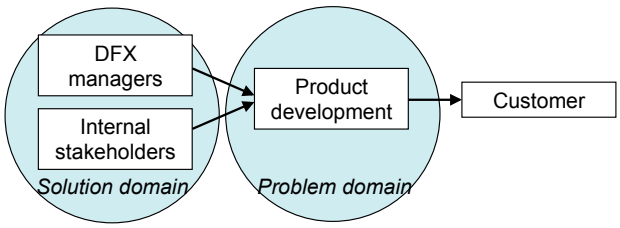

Fig. 7. DfX as a solution domain issue

In this case, the problem domain is the product development organisation of the company, and the solution domain is the internal DfX management organisation that provides knowledge, guidelines, and instructions. Thus, the original solution domain becomes the problem domain. This realisation shows that the concepts of problem domain and solution domain may, and in fact should be, applied recursively and iteratively when needed.

As a practical solution to guide and support the developers and their work, the DfX approach was applied. Behind each DfX discipline, representing different developmental views, is a platform of knowledge and technology. These platforms are knowledge bases that cover both the product and the processes, where requirements, development guidelines, and know-how are managed. DfX disciplines provide solution strategies and best practices for both management and production, they gather and disseminate knowledge and experience, and they build the core competency of the company. DfX management organisation defines and maintains DfX requirements and targets based on agreed-upon platform specifications, while R&D platforms develop and maintain basic solutions and guidelines to be applied in product programs.

Article III shows the advantages of using DfX not only used as a product development tool, but also for management, as it shapes the common vision and harmonises the practices. The article also argued that DfX is useful not only as philosophy, but also as a practice that works through principles and tools, and that it offers a concrete way to manage and coordinate knowledge.

3.4 Article IV: A Case Study of Requirements Management: Toward Transparency in Requirements Management Tools

Article IV shows the importance of transparent processes and tools, thus answering research question 4: *How does transparency support the collaborative work and creation of shared understanding?* The article is based on eleven qualitative interviews in a single company operating in the industrial automation domain. Prototyping was used in addition to the interviews, in order to understand the practicalities better. The company also provided archival material, process and product descriptions, and other related documentation. The results are based on thematic analysis and coding.

While the focus in article IV was on the requirements management phase, the results provide general guidelines that can be applied more widely. It is easy to see the effect that transparent tools and processes have in other phases as well, as

shown in the literature (e.g. Damian *et al*. 2003, Storey *et al*. 2005, Espinosa *et al*. 2007, Jalote-Parmar *et al*. 2010). Transparency enables the developers to be aware of the status of development activities and work items, and it helps to achieve a common, shared understanding of the development goals and objectives. Transparency also assists in achieving effective and open communication, among other desirable effects. Transparency of tools and processes ensures the success of product development.

In the literature discussing the requirements for requirements management, tool and transparency requirements were mostly concerned with the awareness of process and work item states. For example, the areas of decision making, collaboration and communication, and organisation and strategy are often omitted or not addressed extensively in requirements management tool literature. Article IV presented a synthesis of further transparency concerns that should be addressed:

- *Process support*. It is important to have awareness regarding the states and the histories of tasks as well as the characteristic work activities that describe the environment within which they are performed (Omoronyia *et al*. 2010). Transparency and transparent tools enable developers to understand the context of their work, which in turn helps them understand their own goals and relate them to others' goals and work. The main concerns are process states, progress, histories, and context.
- *Tooling and work items*. Awareness support provides information about development artefacts. The main concerns are work artefacts, their states and changes, results, documents, data, and context. It was also considered important to have the ability to link different items to show their dependencies and relationships.
- *Decision making*. Awareness regarding the decision-making process and forums are needed so workers can be aware of the persons who are working on a particular decision (Damian *et al*. 2003). Forums can keep track of decisions, their rationale, and their effects on software products (Aurum & Wohlin 2003). The main concerns are decision-making forums, rationale, the reasoning process, visibility, and documentation.
- *Collaboration and communication*. In distributed development, it is important to know what others' roles and responsibilities are and what they are doing, as it helps to coordinate the collaborative work and diminishes the problem of overlapping work. It is important to understand dependencies of activities and

work items; that is, to have awareness of the other entities that are connected with the one that is being manipulated. This enables individuals to see the impact of their work on that of others (Storey *et al*. 2005). The main concerns are visibility of others' actions, skills and competencies, and information access and exchange.

- *Organisation and strategy*. Development activities and results should be synchronised with portfolios and roadmaps that are based on organisational strategy and goals. For example, Berggren & Bernshteyn (2007, p. 411) suggest "*breaking down the strategy into definitive and meaningful components upon which individual employees can act*". The main concerns are visions, goals, motives, portfolios, and roadmaps. Having awareness of the talent pool was also considered to be useful, especially when planning the work and resource usage.

The transparency requirements presented in article IV are generalised and shown in Table 3.

Table 3. Generalised transparency requirements

Topic	Requirement
Process support	Provide information about the state of the process and tasks
	Show only the task-relevant information
	Have task views that match the actual development tasks
	Provide task guidance
	Provide process guidance
Tooling and work items	Provide information about development artefacts
	Provide standard information templates for work items
	Support linking
	Maintain link validity
	Enforce linking rules among items
	Support traceability
	Support version control
Decision making	Provide the rationale and reasoning process for decisions
	Provide visibility of decisions and their documentation
	Be able to generate status reports from processes
Collaboration and communication	Provide awareness of others' actions
	Provide support for information sharing between management and developers
	Enforce a coherent terminology for work items
Organisation and strategy	Support breaking down the strategy, vision, goals, and motives into work tasks
	Provide information about available resources, skills, and competencies

Detailed descriptions are in the research article. Implementing these requirements helps to address transparency issues, supports human cognition, and enables better decision making and information flow in the development processes. Article IV also shows that transparency helps the development process and improves product quality and development efficiency.

3.5 Article V: Supporting cognitive work in software development workflows

The fifth article answers research question 5: *How is cognitive support provided in workflows?* Action research was used as a research approach, with three action research cycles. During the study, a total of 35 interviews and 36 observations (as described in section 1.4) within one ICT company were analysed using thematic analysis. In addition to interviews and observations, archival material, process and product descriptions, and other relevant documentation were examined.

Article V suggests that having proper support for cognitive work in software development workflows will help developers use available knowledge to come up with creative solutions to non-routine situations, thus improving efficiency and the results of the product creation process. In order to achieve those benefits, cognitive support needs to be integrated into the organisation's workflow. With the ability to provide the necessary information, already identified and tried solutions, and support for problem solving, the workflow could help developers considerably.

This article presents a model designed to support cognitive work and collaboration in software development workflows, as well as its theoretical basis. The model was refined and evaluated in an action research study wherein a tool prototype actualising the model was tested in a real environment. The model (see Fig 8.) consists of three levels that provide complementary views to software development—Why, What, and How—which are the basic questions defining the work.

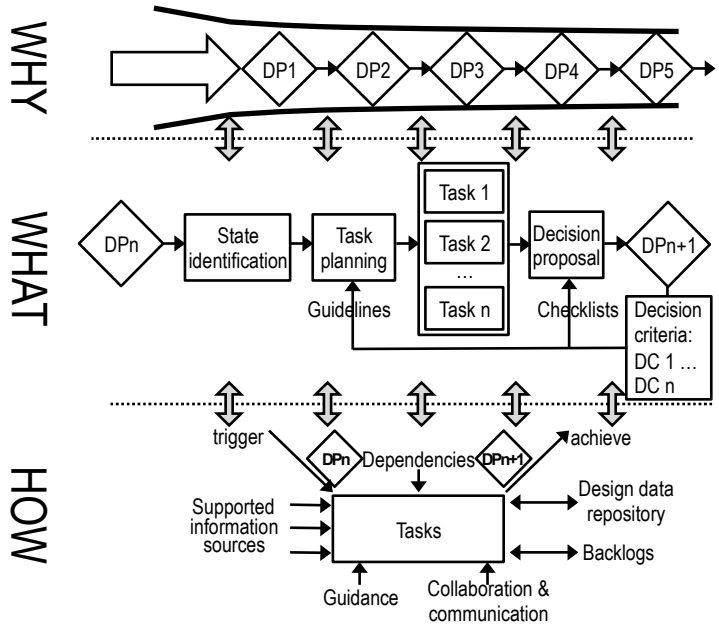

Fig. 8. Three levels of knowledge—three complementary views of development (Hyysalo *et al*. (2013) Publishing permission by Springer)

- *The WHY level* answers the question of why different operations in product development are needed, describes the processes and their objectives, sets the context for development, and defines the criteria for achieving the purposes. The activities in the development process are justified, and the criteria to achieve the purposes, values, and priorities are defined at this level.
- *The WHAT level* splits the development phases into activities and tasks that instantiate the high-level purposes. The order of operations is not strictly defined; instead, the activities are synchronised and coordinated by decision points (DP in the figure). For all activities, there is a defined input that is made available, along with information about tasks, resources, objectives, decision criteria (DC in the figure), and the context. After the task implementation, the results of activities and tasks are presented for decision makers as proposals that are evaluated against decision criteria defined by stakeholders. The decision criteria also provide guidelines for task implementation and provide a checklist for decision makers.

– *The HOW level* focuses on the ways in which the tasks can be accomplished and provides the information needed for work. Task descriptions and guidance for task implementation are provided, as are links to data sources. Developers also accumulate information, knowledge, and experiences during task implementation, which are recorded to maintain the data, even if the member leaves the team.

Utilising this model improves the creation of shared understanding, awareness, and task management, which are aspects that need to be integrated into the company's workflow. Providing an information flow that includes the cognitive information created during the cooperation results in a collaborative working environment that provides transparent, instant, seamless, and flexible collaboration across organisations, teams, and processes.

Awareness and shared understanding must be created for developers to comprehend fully the context and purpose of their work tasks as they relate to the whole product development process, as well as its goals and purposes. A common goal is thus defined towards which the whole development team can aim.

Developers must also be supported with seamless information sharing and awareness about their colleagues and what they are working with, in order to foster an understanding about the situation and the dependencies of their tasks on the tasks of others; i.e., coordination of efforts.

Finally, task management helps developers allocate the resources of the workplace to individual tasks and form communities that work with backlog items.

The main contribution of this article was to complement traditional workflows by providing cognitive support. This support is designed to help developers in their daily tasks, with an emphasis on cognitive work with relevant information and solid reasoning regarding the developers' tasks, awareness support, and the provision of concrete work guidance. The empirical evaluations showed how cognitive support integrated into the workflow is useful, how it improves developers' abilities to accomplish their work, and how the model can even change developers' thinking to better match the intentions and purposes of the processes. The model enhanced the developers' understanding, and with the prototype, the developers were also able to respond to changes and solve practical development problems more efficiently. The quality of the work results improved with the prototype.

3.6 Article VI: Software development as a decision-oriented process

Article VI focused on development processes; the main idea was to show how requirements from different stakeholders guide the development work via decision criteria, thus answering research question 6: *What type of process guides and controls the collaborative development?* This study was based on 46 qualitative interviews in two large companies, operating in distributed mode, in the ICT and automation domains. Both companies also provided archival material, process and product descriptions, and other relevant documentation. All of the material was examined through thematic analysis.

Article VI shows that clearly defined acceptance criteria are needed, both from a process and a product point of view. These criteria guide decision-making activities and define the information content that needs to be created in development tasks. Product-related acceptance criteria are derived from the requirements describing the minimum effort needed to implement a requested artefact. In addition, process-related criteria are a set of predefined rules defining the fulfilment criteria for tasks from the development process point of view, such as relating to input required for subsequent tasks.

It is important that the decision making and decision criteria are integrated in the development process. In this manner, the decisions and decision criteria define the goals for the developers, and the dependencies among goals define the dependencies among activities.

Goals defined by decision criteria are important: for example, Alves & Finkelstein (2002) stated that in requirements engineering, goals provide the rationale and goal refinement provides suitable abstraction levels that support decision makers in evaluating the alternatives.

The whole product design flow provides a common context for developers and managers and supports the creation of shared understanding and transparency of development activities and work items during the development process.

Fig. 9 presents an example of a part of a generic software development process divided into various process elements that build up the process.

Process elements are divided into activities and then further into tasks that work towards the completion of the work. Each element, activity, and task belongs to a certain context. In turn, the whole process is in a context that is affected by the organisation, environment, stakeholders, etc. Situations are reacted to via decision criteria that make up one of the controls defining the boundaries of

acceptable performance. Developers can carry out their tasks within these boundaries.

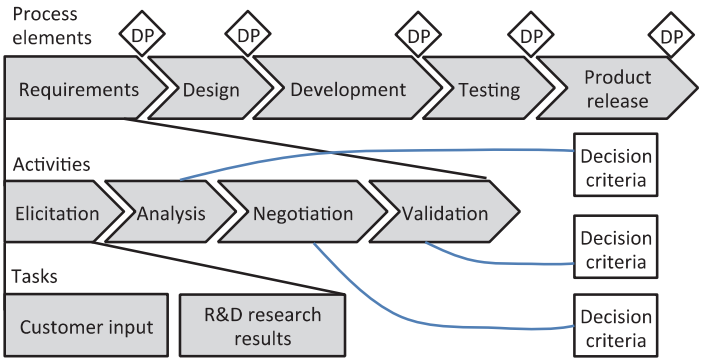

Fig. 9. Example of process showing the context of process elements, activities, and tasks (Hyysalo *et al*. (2014) Publishing permission by Springer)

Acceptance criteria are transformed into decision criteria, which guide and drive the task planning along with the goal(s). Decision criteria are converted into guidelines for both designers and reviewers, and are connected with decision points, as shown in Fig. 10.

Decision proposals are prepared after completing activities consisting of tasks; decision proposals are then sent for approval. The approval decision in a decision point is the final activity, after which an item is ready for the next process element(s).

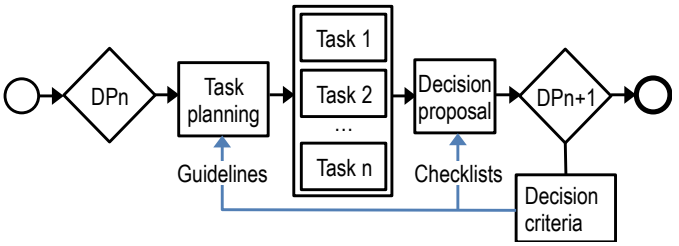

Fig. 10. Decision criteria guiding the development work (Hyysalo *et al*. (2014) Publishing permission by Springer)

In Fig. 11, the arrow on the left points to the input that comes into a process element. An element contains activities, which in turn contain a set of tasks that

must be accomplished to complete a single activity. The process element example has three activities that also have defined decision criteria in a related decision point. All of the information for activities A, B, and C must be fulfilled in order to make a decision and submit the results to other process elements. Therefore, activity describes work that is necessary for creating the required information for a certain decision criterion.

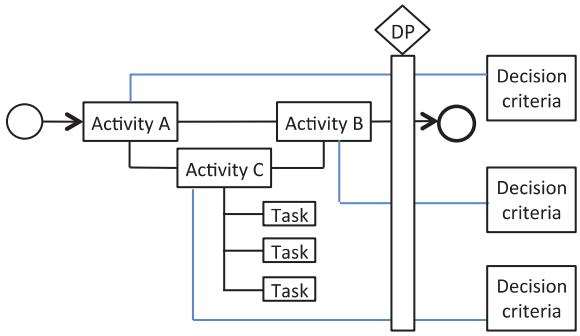

Fig. 11. Example of a process element (Hyysalo *et al*. (2014) Publishing permission by Springer)

The purpose for the links between activities and decision points is to show the relation between these items. For example, activity A in Fig. 11 depends on incoming information as well as information from activities B and C. The picture, however, does not say which activity comes first; it only describes how the activities depend on each other and where the information comes from. Inside an activity, there is a set of tasks necessary to create the information content for the decision criterion. For example, there are three tasks for activity C, and all of their information is defined in the decision criteria for that activity.

Decision points are used to coordinate and synchronise work, and they define the information content the process element produces as output for other process element(s). A decision point includes a list of decision criteria that needs to be fulfilled before the information content produced in the process element can be sent to other process elements. These decision points guide what the actual information content will be, as well as the information needed to make a decision. Decision points not only define what should be achieved, but they also express why it is needed, providing the rationale (explanation) and reason (motive) for the work. Thus, developers know what they are trying to accomplish and why.

In sum, article VI discusses the decision-oriented nature of software development and presents the process in a way that can be adapted easily to different organisations. The aim is to address the identified need for a decision-oriented process, to support collaboration and communication, and to address the decision-making-related issues currently present in development processes. This article describes how the development process works at different abstraction levels, what type of information is needed, and how the information is processed at different levels, and it describes how the information flows. Article VI provides a way to generate the required data utilising available tools and enables management support for process monitoring, decision making, and up-to-date reporting from the process by focusing on the information content and flows.

4 Discussion

In this chapter, the main results and their implications are discussed, as well as the validity and limitations of this study, and finally, future research opportunities are outlined.

4.1 Main implications

The research problem was studied through six research questions discussed in six individual articles. The articles contributed toward the overall research problem, which was stated as:

> *How to provide cognitive support for the collaborative development of embedded systems?*

As stated previously, there are several possible ways to approach a research problem, and in this work, the research problem is addressed from the following six complementary viewpoints: collaboration practices, communication, managing and coordinating knowledge, transparency and awareness, cognitive support, and development processes. Each article provided new ways to support cognitive work in the form of tools, practices, or processes that enable users to understand quickly how the process works, how information flows, the roles of others, and the purposes of each process element, activity, and task. A summary of the main contributions and implications of each article are presented in Table 4.

Table 4. Research questions and implications

Article	Primary research question	Main contributions and implications
I	What are the problems of and solutions to collaborative development?	Documenting challenges and solutions for collaborative development Learning from the embedded systems development industry (success factors, critical areas for improvement, solutions for collaboration)
II	How to support communication in distributed development?	Identifying the importance of communication for collaboration Identifying the importance of awareness and shared understanding for collaboration Documenting communication practices to support collaborative development
III	How to manage and coordinate knowledge in embedded systems development?	Proposal for knowledge management and coordination through the concept of DfX Proposal for harmonisation of different stakeholder views Providing better visibility of information for product development and management Practical example of how to organise DfX in the context of requirements engineering
IV	How does transparency support the collaborative work and creation of shared understanding?	Defining transparency and awareness requirements for tools and processes Providing support for human cognition and decision making through transparency and awareness Enabling developers to understand their work and its context in order to create shared understanding
V	How is cognitive support provided in workflows?	Providing a model to support cognitive work Enabling developers to understand their work and its context in order to facilitate common understanding Example of how to integrate cognition supporting elements into a workflow
VI	What type of process guides and controls collaborative development?	Model for decision-oriented software development Defining the elements of decision-oriented software development Defining information flows in software development

The specific implications of each research article are briefly outlined next.

RQ1: What are the problems of and solutions to collaborative development?

Article I studied the challenges in collaborative software development, both from a literature and industrial point of view, thus complementing the existing literature (see e.g. Olson *et al*. 1998, Herbsleb *et al*. 2001, Herbsleb & Moitra 2001,

Herbsleb & Mockus 2003, Damian & Zowghi 2003, Damian & Lanubile 2004, Paasivaara & Lassenius 2004, Jiménez *et al*. 2009, Noll *et al*. 2010). The article identified the most critical areas for improvement from a practical point of view, i.e. the collaboration issues and needs of the embedded systems industry. Several of the challenges presented in article I can be addressed through cognitive support, which is discussed in articles II–VI included in this dissertation.

RQ2: How to support communication in distributed development?

Article II, which was a follow-up study to article I, focused on communication issues and solutions in the form of practices and tools. Article II examined collaboration and communication through the theories of media richness (Daft & Lengel 1986), media synchronicity (Dennis & Valacich 1999), and social presence (Short *et al*. 1976). These theories discussed the importance of understanding information, as well as how it depends on the media format. Often, the outcome is the centre of the communication. The more uncertain and ambiguous the task is, the richer the media should be, including a degree of awareness. Team members communicate in order to have a shared understanding, and information is exchanged either synchronously or asynchronously. However, technology is often needed, for a variety of reasons. Article II suggests that awareness is critical to collaboration and necessary for a better understanding of the information.

RQ3: How to manage and coordinate knowledge in embedded systems development?

Article III discussed the importance of understanding different stakeholders and their needs. It provided a perspective on knowledge management, with practices and guidelines for development through the concept of DfX, and it promoted aligning the different developmental aspects, taking into account the different stakeholders' views equally, and harmonising the goals. While theoretical and engineering papers discussing DfX are available (see e.g., Bralla 1996, Sheu & Chen 2007, Gehin *et al*. 2008), the new contribution of article III is to outline the way to organise DfX within a company. In contrast to conventional thinking, it is advisable to organise DfX through the organisation that pays the costs of development. Article III also proposes a way to gather, manage, and disseminate knowledge. This article discussed how knowledge obtained from stakeholders forms guidelines and principles, and how that knowledge can be utilised through the use of knowledge bases, at the same time making the different development

perspectives more visible. Following the recommendations will result in a sustainable design, efficient and profitable delivery process, and customer satisfaction throughout the whole product lifecycle.

RQ4: How does transparency support the collaborative work and creation of shared understanding?

Article IV argues that transparency and awareness are among the key knowledge enablers. Awareness is also critical for collaboration (Sarma 2005, Jiménez *et al*. 2009, Omoronyia *et al*. 2010). Technology is needed for creating and supporting transparency and awareness, and those technologies should be designed to support the development work and processes. Transparency and awareness have already been discussed in the literature (see e.g. Beaudouin-Lafon & Karsenty 1992, Grinter 1995, Gutwin *et al*. 1996, Herbsleb 2007, Berggren & Bernshtey, 2007). However, this article shows that there are still requirements that remain unidentified. To fill this gap, article IV provided a set of requirements that the tools and processes must fulfil in order to provide the transparency and awareness necessary for the creation of shared understanding, decision making, communication, and collaboration in general. Most of the identified requirements can be generalised to concern processes, practices, and ways of working as well.

RQ5: How is cognitive support provided in workflows?

Article V draws together the results of articles I–IV and integrates cognitive support into a workflow that structures the work and development process. Workflow is complemented with information flow that also contains cognitive information and support. This type of cognitive support enables the creation of shared understanding and information and knowledge sharing—all of which are requirements for successful collaboration. This work complements traditional workflow models (see e.g. Bracchi & Pernici 1984, Jennings *et al*. 1996, Kwan & Balasubramanian 1997, WFMC 1999, Yu & Schmid 1999, Klein & Dellarocas 2000, van der Aalst & Basten 2002, Adams *et al*. 2006) by providing cognitive support. Article V argues that different supportive assets should be integrated into the company's workflow, thus linking the developers and their tasks to the development process, information, knowledge, and tools.

RQ6: What type of process guides and controls the collaborative development?

Article VI completed the work by defining the elements of a decision-oriented development process and showing how a decision-oriented approach guides and controls the work and defines why the work is done. It discussed how decision points are used to coordinate and synchronise the work and define the information content that is created in development tasks. With the decision-oriented approach, developers are able to address the dynamic development environment of today's software business and the changes that are inevitable. Different abstraction levels of work were identified, and goals and high-level objectives were presented for each of them, along with the justification and rationale for decision making.

In summary, the theoretical implications of this dissertation create new knowledge for companies developing embedded systems in a collaborative environment. The dissertation identifies current challenges and describes potential solutions for providing cognitive support for complex information-intensive knowledge work.

The implications for practice are the experiences and solutions from the embedded systems industry that support collaboration, communication, proposals for knowledge management and coordination, improved transparency and awareness, cognitive support for workflows, and a process model that addresses the decision-oriented nature of software development.

All of the proposed solutions tackle the cognitive challenges discussed in this dissertation. The main implication of this work can be summarised as a need to understand the work and its different abstraction levels fully. This research, as a whole, provides a better understanding for companies developing embedded systems in collaboration regarding how to make their product development more efficient and effective through cognitive support. In this dissertation, support was discussed from different angles; in particular, ways to reduce the cognitive burden of developers was investigated. The main result of the dissertation is a set of solutions integrated into the organisation's workflow to support collaborative development and to help understand the needs of stakeholders. The main idea is that the necessary information is provided, and work and its objectives are justified and put into a correct context. In addition, a development process was outlined that addresses the decision-oriented nature of software development in a way that the necessary data is provided for decision points that guide, coordinate, and synchronise the development efforts. Solutions to cognitive challenges were defined from six complementary viewpoints, each of which was discussed in an individual research article. The common factor in all of these viewpoints is the need to offer developers the needed information and the reason for the work.

4.2 Relevance and validity of the research

Initially, the challenges to be solved by this research came from the practical problems emerging from companies developing software systems in collaboration. The problem relevance was checked against the literature, where a similar gap was identified. Consequently, it is reasonable to assume that the research topics are relevant for the industry, hence increasing their external validity.

The identified issues are relevant for the industry, and the proposals discussed in this dissertation clearly address the needs of the industry. The relevance and practical utility of the construct have already been assessed through a weak market test outlined by Kasanen *et al*. (1993). In a *weak market test*, the construct is applied by the organisation's management; in a *semi-strong market test*, the construct has also been widely adopted by other organisations; and in a *strong market test*, the construct can be shown, systematically, to generate better financial results. At this point, one of the case organisations has invested in the development of a workflow and underlying information systems that implement most of the proposals presented in this dissertation, and the results are already being deployed in selected parts of the organisation. In addition, a second organisation has decided to begin the tool development process, based on case study results. Thus, the construct has clearly passed the weak market test, although it has not yet met the two stronger market tests.

The quality of research can be determined by the reliability and validity of the results, which were obtained through proper research design and methodology. The validity of the research refers to the trustworthiness and the true and unbiased nature of the results. It is necessary to address the validity from the very beginning of the study. Four tests are proposed to establish the quality of empirical research in software engineering: internal and external validity, construct validity, and reliability. (Yin 2009, Wohlin *et al*. 2003, Wohlin *et al*. 2012).

The articles that are the basis of this dissertation underwent a thorough review process, and they were subjected to critical assessment by the scientific community. The research results were open to scrutiny, first by experts within the participating organisations, and then by the profession at large. The validity threats are considered to be under control; the four types of validity threats are addressed as follows.

Internal validity

Internal validity is a concern when causal relations are considered, and the possible disturbing factors indicate a causal relationship, although there is none—whether event A leads to event B, or is there a third factor C that may actually have caused B. It is about knowing all the factors. The internal validity is affected by such influences as how the subjects were selected and divided into different classes, how the subjects were treated and compensated, and whether special events occurred during the experiment. (Yin 2009, Wohlin *et al*. 2012).

Internal validity regarding cause–effect relations was addressed via multiple cases, multiple sources of evidence, and with iterative research gradually building the final outcome. Evaluation of utility, quality, and efficacy was done extensively with the help of industrial experts and real users of developed constructs. Immediate feedback was gathered and the use of prototypes was observed. Based on the rich feedback and analysis, further development and corrective actions were carried out. At the end of each case, a seminar was held to present the results to a wider audience in the case organisations, and feedback from those seminars was incorporated into the development. In addition, each individual study was reported as a conference article in cooperation with other researchers and industrial experts.

External validity

External validity is concerned with the extent to which it is possible to generalise the findings outside the study settings and the extent to which other people beyond the case study find the results interesting. It is affected by the study design and the objects and subjects chosen. (Yin 2009, Wohlin *et al*. 2012).

Dozens of industrial experts have been involved with the study, providing their views, and several organisations, including different types of organisations and different domains, were involved, thus increasing the external validity and generalizability of the results. However, further studies are needed in order to generalise the results further.

Construct validity

Construct validity reflects the extent to which the operational measures studied represent what the researcher had in mind and what was investigated according to

the research questions. Construct validity refers to the relationship between theory and observation. To meet the construct validity, the researcher must (1) select the specific types of issues that are to be studied and (2) demonstrate that the selected measures of these issues actually reflect the specific types of issues that were selected. (Yin 2009, Wohlin *et al*. 2012).

The research problem was viewed from six complementary perspectives, using six conference articles. The research problem and each perspective were also reflected against the existing literature. The main source of empirical data was the industrial experts that were interviewed. The industrial experts had an opportunity to provide feedback on the research and the conclusions that were made based on the interviews. Furthermore, the research data was collected in various ways in order to ensure construct validity. The work was evaluated during development through regular workshops that guided the direction of the work and addressed the problems that emerged. After the constructs were developed, they were evaluated in workshops, or in the case of prototypes, with use in their intended settings. The feedback and evaluation results were used to improve the constructs. However, had different industrial experts been interviewed when defining the research areas to be studied, or had the studied industry been different, the results could have been influenced to some degree. A different selection of perspectives or themes could also influence the obtained results.

Reliability

The objective of reliability is to establish the quality of the research. Reliability is concerned with the extent to which the data and analyses are dependent on the specific researcher. It deals with the ability to draw correct conclusions. (Yin 2009, Wohlin *et al*. 2012).

As described in section 1.4, the research and development of artefacts in this research were conducted according to rigorous, well-defined methodologies and processes. The research process and methodology was documented carefully and presented in further detail in the individual articles, making it possible to repeat the research and compare the findings. Each developed artefact is the result of evolution and was extensively verified in its intended setting by several industrial experts. Each artefact can be used to support collaborative software development. Finally, the compilation part of this dissertation, discussing the overall conclusions, was also documented in a careful manner. However, no researcher is perfect, and therefore, incorrect conclusions are possible.

4.3 Limitations and future research

The main limitation of this study is the long time period, which started in 2005 and ended in 2014. During this time period, a lot has happened, the knowledge base has evolved, and new systems and environments have been developed. This has been taken into account in the study and in its results. Basically, the main problem has been the same over the years—how to provide cognitive support for collaborative software development. This is such a huge problem to be solved that only partial improvements could be provided, and there is still a need for further work, as well as several topics for future research.

Another limitation involves the research methodologies. The studies on cognition, work, and work practices in this research were not always conducted during the actual work situation. For example, the interviews and surveys took place after the work situation. Interviews and surveys are not best way to map the details of work processes, as details may be forgotten or not consciously recognised; observations are a better way to study work processes. However, the phases in which interviews and surveys were used were focused on larger concepts instead of the small details of the work processes, and the methods were used to determine the motivations behind the work. When the actual work processes and practices were the focus of the study, observation was also used as a research method. However, for practical reasons, videotaping was not always possible, and the researchers had to rely on their senses to catch the relevant meanings.

This dissertation thesis and individual articles offer a sound basis for future studies including development of tools and development environments following the recommendations, and using the parts already provided. The first efforts for building tools and a development environment have already been taken in two case companies that continued the work with their internal effort. In addition, the work to make the development efforts publicly available is being planned in the forthcoming AMALTHEA4Public project proposal [1] that continues the development of an integrated development environment with the intention to launch it for public use.

[1] ITEA3 labelled project, funded by European Commission during 2014-2017.

5 Summary

The current turbulent world of software products and their development is complex, which causes several cognitive challenges, as discussed in this dissertation. These challenges, for example, hinder the developers' ability to define common goals, achieve and understand the information, and create a shared understanding of the product and the process of developing it. Therefore, supporting the cognitive work of developers has a significant effect. The main research problem is stated as:

> *How to provide cognitive support for the collaborative development of embedded systems?*

To address this research problem, six studies were carried out to achieve a broad view and understanding regarding the need to support the cognitive work of developers in collaborative embedded systems development. The challenges were studied in more than twenty different companies, including over one hundred interviews. The six articles resulting from the studies formed a logical chain, with interrelated research questions. Each article covers an area that complements the other articles and contributes towards the overall research problem, see Fig. 12. The research questions cover both theoretical and practical levels.

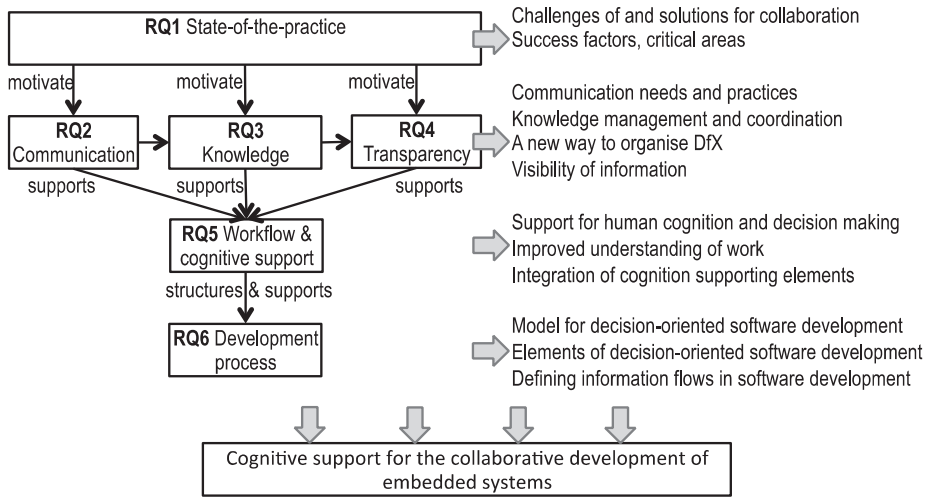

Fig. 12. Summary of the studies and implications

This study and its implications highlight the importance of addressing the different abstraction levels of work and understanding the information flows and needs in order to provide developers clear understanding about what they need to do, why they need to do it, and how.

This dissertation makes an important contribution to software engineering research by providing studies on cognitive issues that focus on supporting collaborative software development. The contributions of this dissertation can benefit both researchers and practitioners by providing a framework that defines a set of tools, methods, and practices that are integrated into a workflow, while the work of developers is linked to processes, information, and knowledge in a work context.

The solutions proposed in this dissertation focus on the following cognitive aspects of collaborative work and processes:

- Transparency and awareness support for tools and processes
- Work context and situation awareness; in particular, defining the purposes, objectives, and roles of the work systems
- Providing common goals and criteria to measure how a system can achieve its purposes, and defining the functions required to achieve the purposes through decision-oriented development
- Knowledge management aspects, such as acquisition, sharing, and utilisation, including also the use of experience and skills
- Fulfilling developers' information needs and defining the information flows to support communication and information distribution
- Proposing how activities and tasks can be implemented, as well as what resources are needed

The industrial cases prove that utilising the suggested solutions improves collaboration among organisations and teams by helping dissemination and use of needed information, especially improving task implementation and decision making. Mitigating the cognitive burden will speed up the development work and reduce the required effort from developers and decision makers. All together, the contributions summarised in this dissertation provide a better understanding of the work and its context for developers and decision makers, and the contributions help increase communication and coordination in collaborative development. The result is better product quality and shorter development times, as the work activities fulfil their purpose more effectively and provide less waste. Finally, by

applying these results, developers can respond to changes and unexpected events, and they can solve development problems in an innovative manner.

References

Abecker A, Dioudis S, van Elst L, Houy C, Legal, M, Mentzas G, Müller S & Papavassiliou G (2002) Enabling workflow-embedded OM access with the DECOR toolkit. Knowledge Management and Organizational Memories: 63–74.

Adams A, Blandford A & Lunt P (2005) Social empowerment and exclusion: A case study on digital libraries. ACM Transactions on Computer-Human Interaction 12(2): 174–200.

Adams M, ter Hofstede A, Edmond D & van der Aalst W (2006) Worklets: A service-oriented implementation of dynamic flexibility in workflows. Proceedings of CoopIS'06: 291–308.

Ahmed T & Tripathi AR (2010) Security policies in distributed CSCW and workflow systems. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 40(6): 1220–1231.

Alavi M & Leidner DE (2001) Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. MIS Quarterly 25(1): 107–136.

Alves C & Finkelstein A (2002) Challenges in COTS decision-making: A goal-driven requirements engineering perspective. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering: 789–794.

Aurum A & Wohlin C (2003) The fundamental nature of requirements engineering activities as a decision-making process. Information and Software Technology 45(14): 945–954.

Babbie ER (1990) Survey research methods. Belmont, CA, Wadsworth.

Basili VR (1993) The experimental paradigm in software engineering. Proceedings of the International Workshop on Experimental Software Engineering Issues: Critical Assessment and Future Directions: 3–12.

Baskerville R & Wood-Harper AT (1996) A critical perspective on action research as a method for information systems research. Journal of Information Technology 11(3): 235–246.

Beaudouin-Lafon M & Karsenty A (1992) Transparency and awareness in a real-time groupware system. Proceedings ACM Symposium on User Interface Software and Technology: 171–181.

Berggren E & Bernshteyn R (2007) Organizational transparency drives company performance. Journal of Management Development 26(5): 411–417.

Birk A & Heller G (2007) Challenges for requirements engineering and management in software product line development. Requirements Engineering: Foundation for Software Quality: 300–305.

Bjørnson FO & Dingsøyr T (2008) Knowledge management in software engineering: A systematic review of studied concepts, findings, and research methods used. Information and Software Technology 50(11): 1055–1068.

Bracchi G & Pernici B (1984) The design requirements of office systems. ACM Transactions on Information Systems 2(2): 151–170.

Bralla JG (1996) Design for excellence. New York, McGraw-Hill.
Braun V & Clarke V (2006) Using thematic analysis in psychology. Qualitative Research in Psychology 3(2): 77–101.
Buckingham Shum S (1998) Negotiating the construction of organizational memories. Information Technology for Knowledge Management. Berlin, Springer Verlag: 55–78.
Carmel E & Agarwal R (2001) Tactical approaches for alleviating distance in global software development. IEEE Software 18(2): 22–29.
Cheng BH & Atlee JM (2007) Research directions in requirements engineering. 2007 Future of Software Engineering: 285–303.
Cooper RG, Edgett SJ & Kleinschmidt EJ (2004) Benchmarking best NPD practices – III. Research Technology Management 47(6): 43–55.
Cruzes DS & Dyba T (2011a) Recommended steps for thematic synthesis in software engineering. International Symposium on Empirical Software Engineering and Measurement (ESEM), 2011: 275–284.
Cruzes DS & Dybå T (2011b) Research synthesis in software engineering: A tertiary study. Information and Software Technology 53(5): 440–455.
Daft RL & Lengel RH (1986) Organizational information requirements, media richness, and structural design. Management Science 32(5): 554–571.
Damian D, Chisan J, Allen P & Corrie B (2003) Awareness meets requirements management: Awareness needs in global software development. Proceedings of the International Workshop on Global Software Development, International Conference on Software Engineering: 7–11.
Damian D & Lanubile F (2004) The 3rd international workshop on global software development. Proceedings of the 26th International Conference on Software Engineering: 756–757.
Damian DE & Zowghi D (2003) RE challenges in multi-site software development organisations. Requirements Engineering 8(3): 149–160.
Dennis AR & Valacich JS (1999) Rethinking media richness: Towards a theory of media synchronicity. Proceedings of the 32nd Annual Hawaii International Conference on System Sciences: 1–10.
Denzin NK & Lincoln YS (2005) Handbook of qualitative research. 3rd Edition. Thousands Oaks, Sage Publications.
Dourish P & Bellotti V (1992) Awareness and coordination in shared workspaces. Proceedings of the 1992 ACM Conference on Computer Supported Cooperative Work: 107–114.
Eskola J & Suoranta J (1998) Johdatus laadulliseen tutkimukseen. Tampere, Vastapaino.
Eskola J & Suoranta J (2008) Johdatus laadulliseen tutkimukseen. Tampere, Vastapaino.
Espinosa JA, Slaughter SA, Kraut RE & Herbsleb JD (2007) Team knowledge and coordination in geographically distributed software development. Journal of Management Information Systems 24(1): 135–169.
Gaines BR (1977) Knowledge management in societies of intelligent adaptive agents. Journal of Intelligent Information Systems 9(3): 277–298.

Gehin A, Zwolinski P & Brissaud D (2008) A tool to implement sustainable end-of-life strategies in the product development phase. Journal of Cleaner Production 16(5): 566–576.

Goel AK (1997) Design, Analogy, and Creativity. IEEE Expert 12(3): 62–70.

Grambow G, Oberhauser R & Reichert M (2011) Towards automatic process-aware coordination in collaborative software engineering. Proceedings of the 6th International Conference on Software and Data Technologies: 5–14.

Grinter RE (1995) Using a configuration management tool to coordinate software development. Proceedings of Conference on Organizational Computing Systems: 168–177.

Grinter RE, Herbsleb JD & Perry DE (1999) The geography of coordination: Dealing with distance in R&D work. Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, 1999: 306–315.

Guindon R (1990) Designing the design process: Exploiting opportunistic thoughts. Human-Computer Interaction 5(2): 304–344.

Gupta A, Pawara KS & Smart P (2007) New product development in the pharmaceutical and telecommunication industries: A comparative study. International Journal of Production Economics 106(1): 41–60.

Gutwin C, Greenberg S & Roseman M (1996) Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation. People and Computers XI: Proceedings of HCI'96: 281–298.

Helo P (2004) Managing agility and productivity in the electronics industry. Industrial Management & Data Systems 104(7): 567–577.

Henninger S (1997) Case-based knowledge management tools for software development. Automated Software Engineering 4(3): 319–340.

Herbsleb JD (2007) Global software engineering: The future of socio-technical coordination. Future of Software Engineering, IEEE Computer Society, 2007: 188–198.

Herbsleb JD, Mockus A, Finholt T & Grinter R (2001) An empirical study of global software development: Distance and speed. Proceedings of the International Conference on Software Engineering, 2001: 81–90.

Herbsleb JD & Mockus A (2003) An empirical study of speed and communication in globally distributed software development. IEEE Transactions on Software Engineering 29(6): 481–494.

Herbsleb JD & Moitra D (2001) Global software development. IEEE Software 18(2): 16–20.

Herbsleb JD, Paulish DJ & Bass M (2005) Global software development at Siemens: Experience from nine projects. Proceedings of the 27th International Conference on Software Engineering: 524-533.

Hirsjärvi S & Huttunen J (1995) Johdatus kasvatustieteeseen. 4th Edition. Helsinki, WSOY.

Hirsjärvi S, Remes P & Sajavaara P (2008) Tutki ja kirjoita. 13th-14th Edition. Helsinki, Tammi.

Hollan J, Hutchins E & Kirsch D (2000) Distributed cognition: Toward a new foundation for human-computer interaction research. ACM Transactions on Computer-Human Interaction 7(2): 174–196.

Hyysalo J, Kelanti M, Lehto J, Kuvaja P & Oivo M (2014) Software development as a decision-oriented process. Software Business. Towards Continuous Value Delivery: 132–147.

Hyysalo J, Lehto J, Aaramaa S & Kelanti M (2013). Supporting cognitive work in software development workflows. Proceedings of Profes 2013, 14th International Conference on Product-Focused Software Process Improvement: 20–34.

Hyysalo J, Parviainen P & Tihinen M (2006). Collaborative embedded systems development: survey of state of the practice. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, 2006: 130–138.

Jacobson I, Booch G & Rumbaugh J (1999) The unified software development process. Reading, MA, Addison-Wesley.

Jalote-Parmar A, Badke-Schaub P, Ali W & Samset E (2010). Cognitive processes as integrative component for developing expert decision-making systems: A workflow centered framework. Journal of Biomedical informatics 43(1): 60–74.

Jennings NR, Faratin P, Johnson MJ, Norman TJ, O'Brien P & Wiegand ME (1996) Agent-based business process management. International Journal of Cooperative Information Systems 5(2-3): 105–130.

Jiao J & Chen CH (2006) Customer requirement management in product development: a review of research issues. Concurrent Engineering 14(3): 173–185.

Jiménez M, Piattini M & Vizcaíno A (2009) Challenges and improvements in distributed software development: A systematic review. Advances in Software Engineering 2009(3).

Kasanen E, Lukka K & Siitonen A (1993) The constructive approach in management accounting research. Journal of Management Accounting Research 5: 243–264.

Klein M & Dellarocas C (2000) A knowledge-based approach to handling exceptions in workflow systems. Computer Supported Cooperative Work 9(3-4): 399–412.

Kulkarni A, Can M & Hartmann B (2012) Collaboratively crowdsourcing workflows with Turkomatic. Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work: 1003–1012.

Kuutti K (1995) Activity theory as a potential framework for human-computer interaction research. Context and Consciousness: Activity Theory and Human Computer Interaction: 17–44.

Kwan MM & Balasubramanian PR (1997) Dynamic workflow management: A framework for modeling workflows. Proceedings of the 30th Annual Hawaii International Conference on System Sciences: 367–376.

Lanubile F, Ebert C, Prikladnicki R & Vizcaíno A (2010) Collaboration tools for global software engineering. IEEE Software 27(2): 52–55.

Lee HL & Billington C (1992) Managing supply chain inventory: Pitfalls and opportunities. Sloan Management Review 33(3): 65–73.

Lehto J, Härkönen J, Haapasalo H, Belt P, Möttönen M & Kuvaja P (2011) Benefits of DfX in requirements engineering. Technology and Investment, 2(1): 27–37.

Mangan P & Sadiq S (2002) On building workflow models for flexible processes. Australian Computer Science Communications 24(2): 103–109.

Marshall C & Rossman GB (1998) Designing qualitative research. 3rd edition, Thousand Oaks, Sage Publications.

Minor M, Bergmann R, Gorg S & Walter K (2011) Reasoning on business processes to support change reuse. CEC 2011: Proceedings of the 13th Conference on Commerce and Enterprise Computing, IEEE: 18–25.

Monasor MJ, Vizcaino A, Piattini M & Caballero I (2010) Preparing students and engineers for global software development: a systematic review. 5th IEEE International Conference on Global Software Engineering (ICGSE), 2010: 177–186.

Myers MD & Newman M (2007) The qualitative interview in IS research: Examining the craft. Information and Organization 17(1): 2–26.

Möttönen M, Härkönen J, Belt P, Haapasalo H & Similä J (2009) Managerial view on design for manufacturing. Industrial Management & Data Systems 109(6): 859–872.

Nakakoji K, Ye Y & Yamamoto Y (2010) Supporting expertise communication in developer-centered collaborative software development environments. Collaborative Software Engineering. Berlin Heidelberg, Springer: 219–236.

Noble D (2004) Knowledge foundations of effective collaboration. Proceedings of 9th International Command and Control Research and Technology Symposium, September 14-16, Copenhagen, Denmark.

Noll J, Beecham S & Richardson I (2010) Global software development and collaboration: barriers and solutions. ACM Inroads 1(3): 66–78.

Olson JS, Covi L, Rocco E, Miller WJ & Allie P (1998) A room of your own: What would it take to help remote groups work as well as collocated groups? CHI 98 Conference Summary on Human Factors in Computing Systems: 279–280.

Omoronyia I, Ferguson J, Roper M & Wood M (2010). A review of awareness in distributed collaborative software engineering. Software: Practice and Experience 40(12): 1107–1133.

Paasivaara M & Lassenius C (2004) Collaboration practices in global inter-organizational software development projects. Software Process: Improvement and Practice 8(4): 183–199.

Pahl G, Beitz W, Feldhusen J, Grote, KH (2007) Engineering design: A systematic approach, 3rd ed. London, Springer.

Patton MQ (2002) Qualitative Research and Evaluation Methods. 3rd edition, Thousand Oaks, Sage Publications.

Pauleen D (2003) Leadership in a global virtual team: An action learning approach. Leadership and Organizational Development Journal 24(3): 153–162.

Pauleen D & Yoong P (2001) Facilitating virtual team relationships via Internet and conventional communication channels. Internet Research: Electronic Networking Applications and Policies 11(3): 190–202.

Pfleeger SL (1995) Experimental design and analysis in software engineering. Annals of Software Engineering 1(1): 219–253.

Robertson T (1997) Cooperative work and lived cognition: A taxonomy of embodied interaction. Fifth European Conference on Computer-Supported Cooperative Work ECSCW '97: 205–220.

Robillard P (1999) The role of knowledge in software development. Communications of the ACM 42(1): 87–92.

Runeson P & Höst M (2009) Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14(2): 131–164.

Runeson P, Höst M, Rainer A & Regnell B (2012) Case study research in software engineering: guidelines and examples. Hoboken, NJ, John Wiley & Sons, Inc.

Sadiq W & Orlowska M (1999) On capturing process requirements of workflow based information systems. Proceedings of the 3rd International Conference on Business Information Systems: 281–294.

Sarker S & Sahay S (2003) Understanding virtual team development: An interpretive study. Journal of the Association for Information Systems 4(1): 1–38.

Sarma A (2005) A survey of collaborative tools in software development. UCI ISR Technical Report, UCI-ISR-05-3. Irvine, University of California, Institute for Software Research.

Sheu DD & Chen DR (2007) Backward design and cross-functional design management. Computers & Industrial Engineering, 53(1): 1–16.

Short J, Williams E, & Christie B (1976) Communication modes and task performance. Readings in Groupware and Computer Supported Cooperative Work. San Francisco, CA, Morgan Kaufmann Publishers, Inc.: 169–176.

Siggelkow N (2007) Persuasion with case studies. Academy of Management Journal 50(1): 20–24.

Storey MA-D, Cubranic S & German DM (2005) On the use of visualization to support awareness of human activities in software development: A survey and a framework. Proceedings ACM Symposium on Software Visualization: 193–202.

Sung JJ (2005) Representation-oriented software development: A cognitive approach to software engineering. Proceedings of the 17th Annual Psychology of Programming Interest Group Workshop (PPIG'05), Brighton, UK: 173–187.

Susman G, Evered R (1978) An assessment of the scientific merits of action research. Administrative Science Quarterly 23(4): 582–603.

Treude C & Storey MA (2012) Work item tagging: Communicating concerns in collaborative software development. IEEE Transactions on Software Engineering, 38(1): 19–34.

Tuomi J & Sarajärvi A (2006) Laadullinen tutkimus ja sisällön analyysi. Helsinki, Tammi.

van der Aalst WMP & Basten T (2002) Inheritance of workflows: An approach to tackling problems related to change. Theoretical Computer Science 270(1-2): 125–203.

van Leijen H & Baets WRJ (2003) A cognitive framework for reengineering knowledge-intensive processes. Proceedings of the 36th Annual Hawaii International Conference on System Sciences: 97–106.

van Merriënboer JJG (1997) Training Complex Cognitive Skills. Englewood Cliffs, NJ, Educational Technology Publications..

Walia GS, Carver J & Philip T (2006) Requirement error abstraction and classification: an empirical study. Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering: 336–345.

Wang M & Wang H (2006) From process logic to business logic: A cognitive approach to business process management. Information and Management 43(2): 179–193.

Weber M & Weisbrod J (2003) Requirements engineering in automotive development: Experiences and challenges. IEEE Software 20(1): 16–24.

Welborn R & Kasten V (2003) The Jericho principle, how companies use strategic collaboration to find new sources of value. Hoboken, NJ, John Wiley & Sons, Inc.

WFMC (1999) Workflow Management Coalition terminology and glossary, Document Number WFMC-TC-1011, Document Status-Issue 3.0. Technical report. Brussels, Workflow Management Coalition.

Whitehead J, Mistrík I, Grundy J & van der Hoek A (2010) Collaborative software engineering: concepts and techniques. Collaborative Software Engineering. Berlin Heidelberg, Springer: 1–30.

Wild C, Maly K, Zhang C, Roberts C, Rosca D & Taylor T (1994) Software Engineering Life Cycle Support – Decision Based Systems Development. Proceedings of the IEEE Region 10's 9th International Conference on Computer Technology, TENCON'94: 781–784.

Wohlin C, Höst M & Henningsson K (2003) Empirical research methods in software engineering. Empirical methods and studies in software engineering: Experiences from ESERNET: 7–23.

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B & Wesslén A (2012) Experimentation in software engineering. Berlin Heidelberg, Springer.

Yin RK (2009) Case study research: Design and methods. Los Angeles, Sage publications, Inc.

Yu L & Schmid BF (1999) A conceptual framework for agent-oriented and role-based workflow modeling. Proceedings of the 1st International Workshop on Agent-Oriented Information Systems.

Zeidler C, Kittl C & Petrovic O (2008) An integrated product development process for mobile software. International Journal of Mobile Communications 6(3): 345–356.

Zhuge H (2003) Workflow- and agent-based cognitive flow management for distributed team cooperation. Information and Management 40(5): 419–429.

Zhuge H, Ma J & Shi XQ (1997) Abstraction and analogy in cognitive space: A software process Model. Information and Software Technology 39(7): 463–468.

Original publications

I Hyysalo J, Parviainen P & Tihinen M (2006) Collaborative embedded systems development: Survey of state of the practice. In Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), March 27-30, 2006, Potsdam, Germany: 130–138.

II Liukkunen K, Lindberg K, Hyysalo J & Markkula J (2010) Supporting collaboration in the geographically distributed work with communication tools in the remote district SME's. In Proceedings of the 5th IEEE International Conference on Global Software Engineering (ICGSE), Princeton, NJ, USA, August 23-26, 2010: 155–164.

III Hyysalo J, Aaramaa S, Similä J, Saukkonen S, Belt P & Lehto J (2009) A new way to organize DFX in a large organization. In Proceedings of Profes 2009, 10th International Conference on Product Focused Software Development and Process Improvement, June 15-17, 2009, Oulu, Finland, Lecture Notes in Business Information Processing, Volume 32: 275–189.

IV Kelanti M, Hyysalo J, Välimäki A, Kuvaja P & Oivo M (2013) A case study of requirements management: Toward transparency in requirements management tools. In Proceedings of the 8th International Conference on Software Engineering Advances (ICSEA 2013), October 27-31, 2013, Venice, Italy: 597–604. ISSN: 2308-4235. ISBN: 978-1-61208-304-9.

V Hyysalo J, Lehto J, Aaramaa S & Kelanti M (2013). Supporting cognitive work in software development workflows. In Proceedings of Profes 2013, 14th International Conference on Product-Focused Software Process Improvement. June 12-14, 2013, Paphos, Cyprus: 20–34.

VI Hyysalo J, Kelanti M, Lehto J, Kuvaja P & Oivo M (2014) Software development as a decision-oriented process. In: Software Business. Towards Continuous Value Delivery: 132–147.

617. Aalto, Esa (2013) Genetic analysis of demography and selection in Lyrate rockcress (*Arabidopsis lyrata*) populations

618. Rodríguez, Pilar (2013) Combining lean thinking and agile software development : how do software-intensive companies use them in practice?

619. Vatka, Emma (2014) Boreal populations facing climatic and habitat changes

620. Isomursu, Marja (2014) Host–parasite interactions of boreal forest grouse and their intestinal helminth parasites

621. Ponnikas, Suvi (2014) Establishing conservation management for avian threatened species

622. Matusek, Florian (2014) Selective privacy protection for video surveillance

623. Virtanen, Elina (2014) Effects of haulm killing and gibberellic acid on seed potato (*Solanum tuberosum L.*) and techniques for micro- and minituber production in northern latitudes

624. Kopatz, Alexander (2014) Genetic structure of the brown bears (*Ursus arctos*) in Northern Europe

625. Loukola, Olli (2014) Information networks among species : adaptations and counter-adaptations in acquiring and hiding information

626. Langrial, Sitwat (2014) Exploring the influence of persuasive reminders and virtual rehearsal on the efficacy of health behavior change support system

627. Jaakkonen, Tuomo (2014) Intra- and interspecific social information use in nest site selection of a cavity-nesting bird community

628. Päätalo, Heli (2014) Stakeholder interactions in cross-functional productization : the case of mobile software development

629. Koskela, Timo (2014) Interaction in asset-based value creation within innovation networks : the case of software industry

630. Stibe, Agnis (2014) Socially influencing systems : persuading people to engage with publicly displayed Twitter-based systems

631. Sutor, Stephan R. (2014) Large-scale high-performance video surveillance

632. Niskanen, Alina (2014) Selection and genetic diversity in the major histocompatibility complex genes of wolves and dogs

633. Tuomikoski, Sari (2014) Utilisation of gasification carbon residues : activation, characterisation and use as an adsorbent

Book orders:
Granum: Virtual book store
http://granum.uta.fi/granum/

UNIVERSITY OF OULU P.O. Box 8000 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

ISBN 978-952-62-0601-1 (Paperback)
ISBN 978-952-62-0602-8 (PDF)
ISSN 0355-3191 (Print)
ISSN 1796-220X (Online)

UNIVERSITY of OULU
OULUN YLIOPISTO