Work in Progress. Expected publishing date June 2007, Information and Software Technology.

Editorial for the Special Issue on Qualitative Software Engineering Research.

Yvonne Dittrich
IT University of Copenhagen
Software Development Group
Rued Langaardsvej 7
2300 Copenhagen S
tel: +45-7218-5177
fax: +45-7218-5000
e-mail: ydi@itu.dk
web: http://www.itu.dk/people/ydi/

Michael John[1]
Fraunhofer Institut for Computer Architecture and Software Technology (FIRST)
Kekuléstr. 7
12489 Berlin
michael.john@first.fraunhofer.de

Janice Singer
NRC Institute for Information Technology
Building M-50, Room 257
Ottawa, ON K1A 0R6
Telephone: +1 (613) 991-6346
Fax: +1 (613) 952-7151
E-mail: Janice Singer
http://iit-iti.nrc-cnrc.gc.ca/personnel/singer_janice_e.html

Bjørnar Tessem
University of Bergen,
Department of Information Science and Media Studies,
N-5020 BERGEN, Norway
Phone: +47 55 58 41 03
Fax: +47 55 58 91 49
Bjornar.Tessem@uib.no
http://www.uib.no/People/sinbt/

---

[1] This work was partially funded by the German Federal Ministry of Education and Research within the project "Virtual Software Engineering Competence Center" (VSEK)

Almost twenty years have passed since the first qualitative research study in software engineering was published [14]. Using qualitative methods and a qualitative analytical framework, Curtis, et al. found communication and cooperation to be critical factors in developing large-scale software systems. Given the importance of this study, it is perhaps surprising that research publications using qualitative methods are still scarce. Therefore, our goal in creating this special issue is to make existing qualitative research more visible and further the understanding of qualitative research and its importance in the software engineering community.

Qualitative research has its main strength in exploring and illuminating the in situ practice of software engineering. This is the everyday practice where software engineers interpret, appropriate, and implement the methods, techniques and processes of the trade. A better understanding of these in situ methods can – in turn – provide a base for their improvement.

From 23 submissions we selected eleven articles. The articles represent a diverse set of theoretical frameworks and methods, while focusing on a wide range of software engineering activities from requirements engineering, project management to software process improvement. The selected articles illustrate the richness of existing research and showcase important and applicable results which will further the discussion of qualitative methods – not only about the concrete results but importantly so, on the value of qualitative research in software engineering in general.

To make the papers more accessible to an audience that may not be used to qualitative research, we introduce this special issue more thoroughly than what is normally seen. By doing so we answer a number of questions related to the practice of qualitative research in the software engineering domain. In turn, each article selected for this special issue addresses some or all of the questions in the context of real research problems.

The editorial proceeds as follows. First qualitative research is introduced and related to the tradition of software engineering. Then we present an overview of the different discourses in which qualitative research on software engineering is published followed by a discussion of potential quality criteria for qualitative research. We end with an introduction to the articles published in this special issue.

What is qualitative research?

There is no 'one way' of doing qualitative research. The only common denominator of qualitative research is that it is based on qualitative data. Some see the possibility to develop an understanding of the software engineering process from a project members' perspective as the main potential of qualitative software engineering research (see especially the paragraphs on applying an ethnographical and ethnomethodological underpinning). Yet qualitative research can come in many different flavours. It can be used under different epistemological paradigms, and with different theoretical underpinnings. It is therefore not possible to talk about *the* contribution of qualitative research as a single entity. To provide a base for understanding and acknowledging the different articles of this volume, we here present some of the many variations of qualitative research.

Qualitative research with a positivistic underpinning might be the most accessible one from a traditional software engineering background: Qualitative researchers, like quantitative researchers, may present their conclusions about the data as objective, truthful statements about the world. For instance, they may state formal propositions about a phenomenon, analyze their data in ways that ensure objectivity, quantify their qualitative data for statistical analysis, and/or argue for the confirmation of hypotheses. An example of an article in this issue that takes a positivistic but still qualitative research approach is the work by Crowston et al. [this volume] where they discuss task assignment mechanisms in Open source software (OSS) projects. Based on a theoretical scheme, Crowston et al. qualitatively code task assignment mechanisms as reflected in email interactions, and conclude (among other things) that self-assignment of tasks is standard in OSS projects, as opposed to in traditional software teams.

Other theoretical underpinnings like structuration theory, Actor-Network Theory, or Activity theory also provide a framework for using qualitative data to understand software engineering practice. Structuration theory, for example, provides three categories of structure that mediate the behaviour of an agent and help to understand the implication of change; interpretive schemes, human and technical resources, and norms. Allison and Merali's article on 'software process improvement as emergent change' [this volume] uses structuration theory as an underpinning in order to understand the influence of existing structures when changing software processes.

Grounded theory [22, 56] supports theory development about the relevant aspects influencing the specificity of a situation, a group of people or companies. The goal is through unbiased analysis of the field material to come to a number of concepts and relationships that determine the topic under research. Grounded theories about the same topic developed by different researchers using different communities as their empirical base can be compared and used to broaden the developed theory. Several of the articles of this special issue apply a grounded theory approach: Rose et al. [this volume] explore management competences of software project managers, Karlsson et al. [this volume] explore the specificities of 'requirements engineering challenges in market driven software development'. Coleman and O'Connor use 'grounded theory to understand software process improvement' [this volume] in an Irish software product company.

Research performed with an ethnomethodological underpinning [13] appears at a first glance to not use nor generate theory at all. Fundamental for a researcher using ethnometodology are the methods the members of a community use to reproduce the social organisation of their common endeavour. Simply said, the researcher is interested in how the participants' themselves make sense of their practices. Communication in the organisation is central in ethnomethodological research: When communicating, members of a group are not only applying their culture and language system, but at the same time reconstructing and developing it. In software development this phenomenon becomes visible in form of project languages which can be understood as local dialects combining professional software engineering terms and domain terms in a unique way. Also the development of organisation specific interpretations and adaptations of methods can be described as a result of such processes. Using an ethnomethdological approach, Rönkkö's article [this volume] discusses how documenting and analysing communication aids in understanding how software engineers interpret, interact and construct their reality.

Ethnographically inspired research [26] focuses on participatory observation and time spent in the field as the main empirical base. Ethnography has been one of the disciplines where the need of qualitative empirical research became visible very early and that contributed centrally to the development of qualitative methodology. The emphasis is on understanding the social rules and norms guiding the behaviour of members of a community from an insiders perspective. In ethnographically inspired researcher, the researcher uses him/herself as an instrument; as the researcher brings his/her expectations into the culture, the unexpected events or clashes with the expected helps the researcher form a better understanding of how the studied culture differs. The learning process by which the researcher 'goes native' reflects his/her growing understanding of this culture, which is generally reported as a rich description by which others can learn and draw helpful conclusions. The article by Robinson et al. [this volume] draws lessons from a set of 'ethnographically-informed empirical studies of software practice' giving an overview of the challenges, possibilities and difficulties of such research. The article by McAvoy and Butler [this volume] reports an ethnographic approach to understanding learning – and what might hinder it – in an agile development team.

In the context of CSCW and information systems design the notion and practice of 'quick and dirty ethnography' [30] has become an established concept, even referred to as a method in the context of requirements elicitation in one of the major textbooks of software engineering [55]. Often such pragmatic approaches start with some concepts derived from literature and further develop them to explore how they unfold in a different environment. In the special issue, Steen [this volume] refers to theoretical discussion of knowledge when exploring the perception and evaluation of 'software product quality' by experienced practitioners. Zannier et al.'s interview study on decision-making in software design [this volume] does not mention any specific underpinning. Also Lutters and Seaman's article on the use of war stories as an interview technique eliciting richer accounts of software practice [this volume] does not put forward specific underpinning. The danger for articles following such a pragmatic approach is that readers – and reviewers – might interpret the result from their own particular perspective and assess the work after criteria which might lead to false expectations and rough judgements.

Qualitative research and software engineering research and practice

The NATO conferences, where the term software engineering was first used, were convened to address the "software crisis" as it was perceived. The contemporary software development practice was seen as problematic; development projects weren't finished on time and budget, and some projects had to be terminated without any result. Unlike other engineering disciplines, software engineering was thus founded as an endeavour to design and develop a practice that was considered lacking.

Perhaps because of this history, it took some time to establish empirical methods in software engineering research, even though the quantitative empirical research paradigm works well within the engineering tradition [23]. The groundwork for empirical methods in software engineering was laid by Victor Basili and the Software Engineering Lab at the University of Maryland with their work on experimentation [4, 5]. The main focus in this type of research is on developing models based on measurable causal relationships

and using them to develop and improve methods for software engineering. A recent example of this kind of research is Hersleb & Mockus' [29] article on an empirical theory of coordination in software engineering.

Parallel to the development of empirical research, software engineering practice has developed: today advanced programming languages and techniques are applied; software engineering process knowledge is widely distributed; and projects are often successful. In this context, qualitative research contributes to the development of knowledge about how methods are interpreted and adapted in specific settings. This knowledge may then be applied to further develop methods and techniques. Qualitative research is particularly well suited to addressing software engineering as a social activity, thus emphasising how software practitioners make sense of methods in their daily work. Such in depth understanding of the social side of software engineering contributes significantly to the development, selection and appropriation of methods, techniques and tools to support the engineering practice.

An appealing property of qualitative research is its potential to involve practitioners in the research process by having them participate in the design, introduction, and evaluation of the improvements. That way qualitative research allows involved companies to address issues that present a problem or challenge for them and this can be discussed and adapted during the research process. (See e.g. Reflective Systems Development approach by Matthiassen [35] or Cooperative Method Development by Dittrich [17].) Such approaches often appropriate action research. Action research is well-discussed within the Information Systems Community. [10] The basic cycle of an action research design is: understanding of the current circumstances; design and introduction of an intervention; which is then further researched. Action research qua definition is not controllable. So even as quantitative evaluation might be applied, the basic cycle always includes qualitative research. The design of interventions aimed at improving the situation of the research subjects is a joint endeavour between researchers and research subjects. All interactions between the researcher, the research subjects and the field are documented and are subject to the analysis. The discussion of possible measures and the appropriation of interventions provide in themselves a source for understanding of the suitability of these measures to the studied problem.

Action research provides a framework to address improvement in software engineering in the context of a qualitative research paradigm and allows for a flexible cooperation with practitioners in industrial settings. One concrete example for the adaptation of action research can be seen in [31]. Here the company and the researchers in collaboration established a different perspective on how to address the requirements the CMM poses on a software development organisation.

The flexibility of the research design in qualitative research thus provides an important advantage for co-operating with industry. Fieldwork methods applied by the researcher can and should be adjusted to the situation at hand. Input from the studied company, like changes in business plans and views on the applicability and appropriation of the research method to the current context are valuable when the researcher assesses the applicability of a research method.

Qualitative research: distributed over different research discourses

Qualitative research publications in traditional software engineering venues are scarce, though they exist. Two workshops at the International Conference of Software Engineering – one in 2000 [53] and one in 2005 [33] – indicate that qualitative research methods attracted interest over a longer period. However, much has been published in other related discourses. Below, we present a brief synopsis of qualitative research on software engineering from the Software Engineering, Computer Supported Cooperative Work, Information Systems, and the agile development approach. Such a brief overview cannot be complete. However, the aim is to provide interested readers with a starting point for their own investigations.[2]

Qualitative Research in the Software Engineering discourse.

Publications within software engineering often combine qualitative and quantitative approaches. Seaman, for example, proposed to complement quantitative methods with qualitative ones. [48] The article proposes to use qualitative research methods as a way to generate hypotheses that then can be tested quantitatively. As the goal is to understand the mechanics of software engineering (i.e., the influence of the deployment of certain methods on the outcome of software engineering) the result focuses on identifying quantifiable causal relationships. Seaman reports here from qualitative observations of inspection meetings that are coded in a quantifiable way so that statistics can be applied.

Other research published in software engineering triangulates qualitative results with quantitative methods. One of the means for quality assurance of qualitative empirical research is the deployment of different methods and data sources or the cooperation of different researchers during the analysis to counter possible individual or methodological biases. Singer et al. [54] provide an example for this kind of triangulation, results from questionnaires, participatory observations, interviews and tool usage statistics were combined in order to understand work practices of software maintenance and compiling a list of requirements for a software exploration tool.

In the past, relatively little research in the software engineering domain presents purely qualitative research. To give a few examples: Herbsleb and Grinter [28] report a study of coordination in distributed development based on interview data and document analysis. Dittrich and Lindeberg [19] report how a traditional process model was used as a frame for iterative implementation and user-developer cooperation. Similarly, Hall and her colleagues have conducted a number of studies using qualitative data that look at a number of different factors involved in implementing software process initiatives (see for example [1, 40]). The requirements engineering subfield of software engineering has always been more attentive to qualitative methods. One good example of qualitative research from requirements engineering is Damian et al.'s [15] work on the impact of requirements on the development process.

The use of qualitative data in software engineering is becoming more prominent. As an example, in his Ph.D., Stillito used qualitative methods to characterize the questions programmers ask during software evolution tasks [52]. Similarly, in his Ph.D., de Souza conducted two field studies to understand how software developers manage the effect of

[2] This section extends a similar section from [18].

dependencies in the coordination of their work. Both of these theses represent recent trends in trying to understand more deeply the social side of software engineering [16]. Furthermore, several recent articles in the IEEE Transactions on Software Engineering have showcased qualitative methods in their research (e.g., [32, 42]) further showing the need for the software engineering domain to utilize qualitative methods to understand many of the deep underlying questions regarding how software is developed and maintained.

Software Engineering as Coooperative Work

As software engineering is a co-operative effort, software engineering has become a subject of discussion in Computer Supported Cooperative Work. [47] Traditionally empirical research in CSCW mainly uses qualitative methods including especially ethnography, and ethnomethodologically informed ethnography, often combined with participatory design processes reported in an action research manner.

Issues addressed include the use of representations and design work as embodied practice [57], organizational constraints and their influence on the work practice [8, 9], the development of organizational patterns from within the project group, [39, 41, 58] and the interaction of work practice and computer based tools [24, 25]. The research focus here is on understanding the ways the members of a software engineering project achieve coordination of their cooperative effort. A more recent example of this kind of research is [46] where the authors explore how software engineers used plans to coordinate a widely distributed method and tool development project.

A number of articles at the most recent CSCW conference (2006) featured software engineering as their domain of study. Additionally, a workshop on the social side of large-scale software engineering featured a number of articles utilizing qualitative methods (proceedings can be downloaded at: http://lizzy.iit.nrc.ca/social_se2006/)

Research in CSCW is primarily rooted in ethnography. The researcher tries to understand software engineering as work practice from within. This sometimes leads to what, for software engineers, looks like the appreciation of a skillfully performed bad practice. The interesting question – what makes a seemingly disadvantageous practice less troublesome than changing the habit – is seldom asked by software engineering researchers. In CSCW research, few authors use the studies to further develop methods or tools for developers. Here the work of Grinter [24, 25] provides an exception.

'Out of Scandinavia'

In reference to the seminal introduction of what non-Scandinavians call the Scandinavian Schools of Systems Development to the international research community (Floyd et al. 1989), this section presents a regionally rooted strand of research, which is primarily published in information systems venues. As the Scandinavian School on Systems Development is characterised by a humble attitude towards the expertise of the future users of the software under development, this strand of research takes the experience of the practitioners as a starting point. [2, 3, 35, 36]

Research is performed as Action Research, in this approach meaning that researchers take part in industry in software development or software process improvement. The active

participation is complemented with qualitative and quantitative data collection, and detailed documentation of the intervention. The data collected provides the basis of the research focusing on the evaluation of the introduced measures. In [36] a major project on software process improvement is reported which involved researchers from different Danish universities and a number of industrial partners. This project resulted in a number of publications addressing the social and organizational conditions for software processes and their improvements (e.g. [37])).

Due to the connection to the Scandinavian approach to systems development, the relation between user participation and software engineering methods and processes is a continuous thread of discussion within this community. As early as 1993, Bansler and Bødker discussed the adequacy of methods to support user-developer communication. [2]. Nørbjerg and Kraft [38] addresses the constraints that software methods and processes put on developers regarding the possibility to take usability into account. [27] reports on a case where agility in the development processes allows developers to quickly react to customer feedback during product development. In 2004 a special issue of the Scandinavian Journal of Information Systems addressed among other interesting research issues, the implications of a changes of the relationship between use and development of web based systems. [7]

Agile development

The agile development community has continuously been claiming the value of sound collaboration and communication practices. The Agile Manifesto (http://agilemanifesto.org) states this is in an almost religious manner. This perspective also dominates Beck's book on extreme programming (XP) [6] as well as Cockburns's book on agile software development [12].

One might expect that the emphasis on collaboration would lead to rich corpus on the social aspects of the agile development processes. However, this does not seem to be the case as researchers in agile methods mainly seem to follow the traditional software engineering community in their choice of topics to study and the empirical methods to apply.

Among the exceptions is a study of pair programming by Williams [60] where the efficiency of pair programming is researched, with qualitative data from interviews with developers serving to elucidate important factors that influence the effectiveness of pair programming. A purely qualitative approach is described in a special issue of the European Journal of Information Systems on agile organisations, where Fitzgerald et al. [21] present a longitudinal case study describing the introduction of agile methods in an existing software development organisation over a period of 2-3 years. Here, semi-structured interviews with managers and key staff in the process were collected to study how local tailoring of agile methods worked out. Another work is Mackenzie et al. [34] who view XP from a CSCW perspective. They explain how the practices of XP can be seen as collective versions of the programmer's daily work habits. Significant is also work by Sharp and Robinson [49] which reports an ethnographic study of XP practice as a culture that is based on and supported by the XP practices but also within the team shows high levels of self-management, self-organisation, and shared responsibility, as well as a relaxed work environment, which is more than what can be explained with

'implementing' a set of practices. All these studies focus on understanding how software engineers make development work, rather than identifying dependent and independent variables and relationships between them.

The main conferences for agile development, the European XP conferences, and the North American XP Universe/Agile Universe/Agile conferences have contained a few presentations of qualitative research. A series of conference articles from Robinson and Sharp (and co-authors) are good examples of qualitative research presented in these conferences [43, 44, 45, 50, 51] as well as a couple of North American contributions from Zannier and Maurer [61] and Chong [11].

Quality criteria for qualitative software engineering research

A major surprise for us as editors was the diversity of opinions expressed during the review process. No single paper achieved consistent reviews, even consistently poor reviews. All reviews of all papers were mixed, with some reviewers praising the paper(s) and others having rather opposite and harsh opinions. This occurred despite a careful selection of reviewers – we had been eager to assign reviewers that were both knowledgeable regarding qualitative research, and known as constructive and competent reviewers. So what is the problem with quality criteria for qualitative research?

Some of the diverging opinions in the evaluations can be traced to differences in the reviewers' disciplinary background. Where one reviewer saw something lacking, such as specific pieces of related work, another saw a decent piece of empirical research. Qualitative research might be subject to particular difficulties in this respect: related research is often scattered across several disciplines and, on top of that, qualitative research is not as controllable as fixed or quantitative designs. So, although the initial questions may not be answered, interesting and important results may still be achieved.

Another issue concerns the epistemological diversity of qualitative research. The section above on 'What is qualitative research' indicates that there is ample opportunity for disagreement on the what and how of qualitative research.

The diverging evaluation thus indicates the importance of this special issue as the presentation and discussion of a number of different articles should not only contribute to developing a common understanding of qualitative empirical software engineering research, but also take the first steps in developing a common way to evaluate the quality of qualitative research.

Based on our experience we propose the following criteria:

- clarity of contribution and research question.
- clarity of the theoretical foundation, that is role of theory and epistemological paradigm applied in the research and the analysis.
- adherence of the chosen research design and adherence of the data analysis to the proposed theoretical foundation.
- discussion of research design and its development throughout the research process.
- quality of empirical work. (How well is the research performed and analysed? Are different data sources used to triangulate the primary field material?)

- presentation of the field material and its analysis. (Can the analysis be followed?)
- reflection on the lessons learned and the applicability and usefulness of the chosen research design.
- reference to related research regarding the claimed contribution.

These quality criteria have to be both accessible and assessable when reading an article. This is the reason for the often discussed and sometimes criticised verbosity of qualitative research. Thick descriptions are essential to assess the quality of fieldwork and analysis.

The articles of the special issue

Due to the above-mentioned diversity in the evaluation by the assigned reviewers, the editorial board discussed every article individually. Each editor then took responsibility for a number of selected articles and corresponded with the authors regarding the necessary improvements to the article. Our end result is a number of high quality contributions representing a broad spectrum of qualitative research:

The lead article by Robinson, Segal and Sharp draws upon a series of ethnographically-informed qualitative studies. Based on their experience, they discuss how a qualitative approach influences the kind of research question that can be addressed, sum up methodological and practical challenges and discuss the nature of their research findings and their impact on collaborators and in the community.

McAvoy and Butler present in their article 'The impact of the Abilene paradox on double-loop learning in an agile team' an ethnographical study of agile development. The Abilene paradox is referred to to make sense of the observation that the team did not use user stories despite the fact that they supported their use.

'Self-organization of teams for free/libre open source software development' by Crowston, Li, Wei, Eseryel and Howison explores the coordination of work tasks in an open source project by analysing the communication via e-mail and groupware tools available online.

The article 'Revealing actual documentation usage in software maintenance through war stories' by Lutters and Seaman proposes an innovative interview technique to reveal rich accounts in order to explore even unanticipated aspects of the interview subject. They demonstrate the technique via analysis of examples of interviews on software maintenance.

The following five articles are interview studies that perform and analyse interviews in very different ways:

Karlsson, Dahlstedt, Regnell, Natt och Dag, and Persson interview practitioners from companies developing software products. They combine interviews with practitioners from different companies and focus group meetings and analyse their material using a grounded theory approach. The article summarises the results regarding 'Requirements Engineering Challenges in Market-Driven Software Development.'

'Management competences, not tools and techniques: a grounded examination of software project management at WM-data' by Rose, Pedersen, Hosbond and Kræmmergaard combines interview data from a number of project managers in one company and with focus group data and uses grounded theory as an epistemological

underpinning. As the research question is a different from above, the results and the presentation of the studies differs as well. Placing both examples side by side gives an indication of the variety of ways to perform, analyse and present grounded theory based studies.

Steen's article 'Practical knowledge and its importance for software product quality' provides a third example of studies based on interviews with practitioners from different companies, this time developing and appropriating philosophical concepts around knowledge as a theoretical underpinning for the analysis.

Zannier, Chiasson and Maurer's article developing 'A model of design decision making based on empirical results of interviews with software designers' provides a description of their analysis process and the conclusions they draw in a detailed way. The strength of the article is in the large number of interviewees from different companies and the development of a high level model of design decision making.

Coleman and O'Connors article 'Using grounded theory to understand software process improvement: a case study of Irish software product companies' uses interviews with a number of companies and grounded theory for their research. Here as well as in the following article software process improvement is subject of the research which allows readers to compare results using different field work methods and theoretical underpinnings within the same research subject.

Allison and Meraly base their article 'Software process improvement as emergent change: a structurational analysis' on the analysis of historical documents, participatory observation over a year and a number of formal and informal interviews from one single company. They analyse the software process improvement over a period of more then 10 years informed by structuration theory. The results show how the outcome of software process improvement endeavours is affected by the existing context and tradition of an organisation.

The final article 'Interpretation, interaction and reality construction in software engineering: an explanatory model' by Kari Rönkkö proposes the adaptation of the 'documentary method of interpretation' from ethnomethodology in order to understand the communication and coordination problems in software engineering and uses fieldwork examples to illustrate the usefulness. This article, as in the previous one, points to the development and use of theoretical concepts to understand software engineering practice.

Enjoy the reading!

Yvonne Dittrich

Michael John

Janice Singer

Bjørnar Tessem

Acknowledgements

Thanks to the following researchers who contributed to the special issue by reviews. Without their contribution the special issue would not have been possible.

- Pär Ågerfalk, University of Limerick, Ireland
- Ian Allison, Nottingham Trent University, UK
- Liam Bannon, University of Limerick, Ireland
- Wolf-Gideon Bleek, University of Hamburg, Germany
- Johanna Bragge, Helsinki School of Economics, Finland
- Keld Bødker, Roskilde University Centre, Denmark
- Gianmarco Campagnolo, University of Trento, Italy
- Sebastien Cherry, École Polytechnique de Montréal, Canada
- Kevin Crowston, University of Syracuse, USA
- Åsa Dahlstedt, University of Skövde, Sweden
- Björn Decker, Fraunhofer Institute for Experimental Software Engineering (IESE), Germany
- Alain Désilets, National Research Council , Canada
- Riitta Sisko Hekkala, Oulu University, Finland
- Ola Henfridsson, Viktoria Institute, Sweden
- Christian Hoecht, Technical University of Kaiserslautern, Germany
- Helena Holmström, University of Limerick, Ireland, and IT Unversity of Göteborg, Sweden
- Harald Holz, German Research Center for Artificial Intelligence GmbH, Germany
- Jens Henrik Hosbond, Aalborg University, Denmark
- Jon Iden, University of Bergen, Norway
- Letizia Jaccheri, Norwegian University of Science and Technology, Norway
- Lena Karlsson, Lund University, Sweden
- Daniel Karlstrom, Lund University, Sweden
- Steinar Kristoffersen, University of Oslo, Norway
- Philippe Kruchten, University of British Columbia, Canada
- Timothy C Lethbridge, University of Ottawa, Canada
- Wayne Lutters, University of Maryland Baltimore County, USA
- John Mcavoy, University College Cork, Ireland
- Hilkka Merisalo-Rantanen, Helsinki School of Economics, Finland
- Nils Brede Moe, The Foundation for Scientific and Industrial Research at the Norwegian Institute of Technology (SINTEF), Norway
- Eric Monteiro, Norwegian University of Science and Technology, Norway
- Anders Morch, University of Oslo, Norway
- Nannette Napier, Georgia State University, USA
- Jacob Nørbjerg, Copenhagen Business School, Denmark
- Rory O'Connor, Dublin City University, Ireland

References

1. N. Baddoo, T. Hall, Motivators of Software Process Improvement: an analysis of practitioners' views, Journal of Systems and Software 62 (2002) 85-96.
2. J. P. Bansler, K. Bødker, A Reappraisal of Structured Analysis: Design in an Organizational Context, ACM Transaction on Information Systems, 11 (1993) 165-193.
3. J. Bansler, E. Havn, The nature of Software Work. Systems Development as Labour Process, in: P. v. d. Besselaar. et al., Information System Work and Organisation Design, Elsevier Science Publication, 1991, pp 145-153.
4. V. Basili, The role of experimentation in software engineering: past, current, and future, Proceedings of the ICSE '96, 1996, pp. 442-449.
5. V. Basili, S. Green, Software Process Evolution at the SEL, IEEE Software 1994 58-66.
6. K. Beck, C. Andres, Extreme Programming Explained: Embrace Change (2nd edition), Addison-Wesley 2005.
7. K. Bødker, P. Carstensen, Development and Use of Web-Based Information Systems, Scandinavian Journal of Information Systems 16 (2004) 3-10.

8. G. Button, W. Sharrock, Occasioned practice in the work of software engineers, in: M. Jirotka, J. and Goguen, Requirements Engineering. Social and Technical Issues, London, 1994.
9. G. Button, W. Sharrock, Project Work: The Organization of Collaborative Design and Development in Software Engineering, Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design, 5 (1996) 368-386.
10. P. Checkland, S. Holwell Action Research: Its Nature and Validity, Systemic Practice and Action Research 11 (1998).
11. J. Chong, Social behaviors on XP and non-XP teams: a comparative study, Proceedings of the Agile Conference, 2005,, IEEE 2005, pp. 39-48.
12. A. Cockburn, Agile software development, Addison-Wesley, Boston, MA, 2002.
13. A. Coulon, Ethnolmethodology, Sage, Thousand Oaks, 1995.
14. A. Curtis, H. Krasner, N. Iscoe, A field study of the software design process for large systems, Communications of the ACM 31 (1988), 1268-1287.
15. D. Damian, D. Zowghi, L. Vaidyanathasamy, Y. Pal, An industrial case study of immediate benefits of requirements engineering process improvement at the Australian Center for Unisys Software, Empirical Software Engineering, 9(2004) 45-75,
16. A. De Souza, On the Relationship between Software Dependencies and Coordination: Field Studies and Tool Support, Ph.D. dissertation, Donald Bren School of Information and Computer Sciences, University of California, Irvine, Irvine, CA, USA, 2005. Available at: http://www2.ufpa.br/cdesouza/publications.html
17. Y. Dittrich, Doing Empirical Research in Software Engineering – finding a path between understanding, intervention and method development, in: Y. Dittrich, C. Floyd, R. Klischewski (eds.), Social thinking – Software practice. MIT Press, 2002, pp. 243-262.
18. Y. Dittrich, Researching the Social Side of Software Engineering, Upgrade 2006.
19. Y. Dittrich, O. Lindeberg, How use-orientated development can take place, Information and Software Technology 46 (2004) 603-617.
20. C. Floyd, W.M. Mehl, F.-M. Reisin, G. Schmidt, G. Wolf, Out of Scandinavia: Alternative Software Design and Development in Scandinavia, Journal for Human Computer Interaction4 (1989) 253-380.
21. B. Fitzgerald, G. Hartnett, K. Conboy, Customising agile methods to software practices at Intel Shannon, Special issue on business agility and diffusion of information technology of the European Journal of Information Systems 15 (2006), pp. 200-213.
22. B. Glaser, A.L. Strauss, The Discovery of Grounded Theory, New Yorck: Aldin, 1967.
23. R.L. Glass, The Software Research Crisis, IEEE Software 1994 42-47.

24. R.E. Grinter, Supporting Articulation Work using Software Configuration Management Systems, Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design 5 (1996) 447-465.
25. R.E. Grinter, Recomposition: Coordinating a Web of Software Dependencies, Computer Supported Cooperative Work 12 (2003) 297-327.
26. M. Hammersley, P. Atkinson, Ethnography Principles and Practices, Routledge London, 2005.
27. C. Hansson, Y. Dittrich, D. Randall, Agile Processes Enhancing User Participation for Small Providers of Off-the-Shelf Software, in: Extreme Programming and Agile Processes in Software Engineering. Proceedings of the 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004.
28. J.D. Herbsleb, R.E. Grinter, Splitting the Organizzation and integrating the Code: Conway's Law Revisited, Proceedings of the 21st International Conference on Software Engineering, Los Angelos, CA, 1999, pp. 85-95.
29. J.D. Herbsleb, A. Mockus, Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering, in: Proceedings, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, Helsinki, Finland, September 1-5 2003, 112-121.
30. J. Hughes, V. King, T. Rodden, H. Andersen, Moving out from the control room: ethnography in system design, in Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work (Chapel Hill, North Carolina, United States, October 22 - 26, 1994). CSCW '94. ACM Press, New York, NY, 1994, 429-439.
31. J. K. Iversen, P.A. Nielsen, J. Norbjerg, Problem Diagnosis Software Process Improvement, in: T. J. Larsen, L. Levine, J.I. DeGross (eds.) Information systems: Current Issues and Future Changes, IFIP 1998.
32. A.J. Ko, B.A. Myers, M. Coblenz, H.H. Aung, An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks, IEEE Transactions on Software Engineering, 32(12), 971-987.
33. M. John, F. Maurer, B. Tessem, Human and social factors of Software Engineering – Workshop Summary HSSE05, ACM SIGSOFT Software Engineering Notes, Vol. 30, Issue 4 (June) 2005, 1-6.
34. A. MacKenzie, S. Monk, From Cards to Code: How Extreme Programming Re-Embodies Programming as s Collective Practice, Computer Supported Cooperative Work 13 (2004) 91-117.
35. L. Mathiassen, Reflective Systems Development. Scandinavian Journal of Information Systems 10(1998) 67-118.
36. L. Mathiassen, J. Pries Heje, O. Ngwenyama, Improving Software Organizations, Addison-Wesley, 2002.

37. P. A. Nielsen, J. Nørbjerg, Assessing Software Processes: Low Maturity or Sensible Practice, Scandinavian Journal of Information Systems 13(2001) 23-36.
38. J. Nørbjerg, P. Kraft, Software Practice is Social Practice, in: Y. Dittrich, C. Floyd, R. Klischewski (eds.), Social thinking – Software practice, MIT Press, 2002, pp. 205-212.
39. C. Potts, L. Catledge, Collaborative Conceptual Design: A Large Software Project Case Study, Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design 5 (1996) 415-445.
40. A. Rainer, T. Hall, Key success factors for implementing software process improvement: a maturity-based analysis, Journal of Systems and Software 62 (2002 71-84
41. T. Robertson, Embodied Action in Time and Place: The Cooperative Design of a Multimedia, Educational Computer Game, Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design 5 (1996) 341-367.
42. M.P. Robillard, W. Coelho, G.C. Murphy, How Effective Developers Investigate Source Code: An Exploratory Study, IEEE Transactions on Software Engineering 30 (2004) pp. 889-903.
43. H. Robinson, H. Sharp, The characteristics of XP teams, in: Proceedings of XP2004 Germany, June 2004, pp. 139-147.
44. H. Robinson, H. Sharp, The social side of technical practices, in Proceedings of XP2005, LNCS 3556 (2005), pp. 100-108.
45. Robinson, H. and Sharp, H. (2005b) Organisational culture and XP: three case studies, in: Proceedings of Agile 2005, IEEE Computer Press 2005, pp. 49-58.
46. K. Rönkkö, Y. Dittrich, D. Randall, When Plans do not Work Out: How Plans are Used in Software Development Projects, Journal of Computer Supported Cooperative Work 14 (2005) 433-468.
47. K. Schmidt, W. Sharrock, Computer Supported Cooperative Work, Special Issue on Studies of Cooperative Design 5 (1996).
48. C. Seaman, Qualitative Methods in Empirical Studies of Software Engineering, in IEEE Transactions on Software Engineering 25(1999) 557-572.
49. H. Sharp, H. Robinson, An Ethnographic Study of XP Practices, Empirical Software Engineering 9 (2004) 353-375.
50. H. Sharp, H. Robinson, A Distributed Cognition Account of Mature XP Teams, in P. Abrahamsson, M. Marchesi, G. Succi (eds.), XP 2006, LNCS 4044(2006), pp. 1 – 10.
51. H. Sharp, H. Robinson, J. Segal, D. Furniss, The role of story cards and the wall in XP teams: a distributed cognition perspective, Proceedings of the Agile Conference 2006, pp. 65-75.
52. J. Sillito, G. Murphy, K. De Volder, Questions Programmers Ask During Software Evolution Tasks, SIGSOFT/FSE 2006.

53. S. Sim, J. Singer, M. Storey, Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research, Proceedings of Empirical Software Engineering 6 (2001), 85-93.
54. J. Singer, T. Lethbridge, N. Vinson, N. Anquetil, An examination of software engineering work practices. Proc. CASCON. IBM Toronto, 209Y223, October 1997.
55. I. Sommerville, Software Engineering, 8th edition, Addison Wesley 2007.
56. A. Strauss, J. Corbin, Grounded Theory in Practice, London: Sage, 1997.
57. L. Suchman, R. Trigg, Artificial Intelligence as craftwork, in: S. Chaiklin, J. Lave, Understanding Practices – Perspectives on Activity and Context. Cambridge University Press, NY, 1993, pp. 144-178.
58. H. Tellioglu, I. Wagner, Negotiating Boundaries. Configuration Management in Software Development Teams, Computer Supported Cooperative Work 6 (1997), 251-274
59. J.E. Tomayko, O. Hazzan, Human Aspects of Software Engineering, Charles River Media, Inc., Hingham, Massachusetts, 2004.
60. L.A. Williams, The Collaborative Software Process, PhD thesis, Dept. of Computer Science, University of Utah, 2000.
61. C. Zannier, F. Maurer, Foundations of Agile Decision making from Agile Mentors and Developers, n P. Abrahamsson, M. Marchesi, G. Succi (eds.), XP 2006, LNCS 4044 (2006), pp. 11 – 20.