Abstract  The development of methods tools and process improvements is best to be based on the understanding of the development practice to be supported. Qualitative research has been proposed as a method for understanding the social and cooperative aspects of software development. However, qualitative research is not easily combined with the improvement orientation of an engineering discipline. During the last 6 years, we have applied an approach we call ‘cooperative method development’, which combines qualitative social science fieldwork, with problem-oriented method, technique and process improvement. The action research based approach focusing on shop floor software development practices allows an understanding of how contextual contingencies influence the deployment and applicability of methods, processes and techniques. This article summarizes the experiences and discusses the further development of this approach based on several research projects in cooperation with industrial partners.

Keywords Empirical research · Research methodology · Cooperative method development · Human factors · Software engineering
1 Introduction

Based on the assumption that software engineering is heavily influenced by its social contingencies, we have during the last 6 years combined qualitative social science research with method improvement in cooperation with industrial practitioners, using action research—a social science research approach combining understanding and improvement of social relations as a methodological reference. The initial formulation of our approach called cooperative method development (CMD) was published in 2002. (Dittrich 2002) Since then, we have applied and refined this approach in a number of projects. Our original approach, based on five guidelines presented in Section 2, has been confirmed as a framework to be adapted to the specific research projects. In the different projects we tried out and developed additional techniques and method elements that are intended to extend the basic framework to be adapted likewise to the specific research context. This article summarises the lessons learned regarding the methodological approach and presents different methodological elements that complement the original framework.

Experiences from the individual projects have—we will argue—led to three groups of results on a methodological level. First, qualitative research can be combined with the improvement of methods, processes and techniques. We will show this in Section 4 by short summaries of the projects and references to published results. Second, qualitative research addresses the role of the communication and cooperation that is necessary to make software development methods work and to situate the development into a specific context. This aspect is discussed in Section 5.3 of the discussion. The interaction between method deployment and even the wider context of development becomes visible. Third, we also propose complementing our initial approach. This is discussed in Section 5 ‘Cooperative Method Development Revisited.’

Some of the earliest requests for taking the social and cooperative aspects of software development into serious consideration were published in the mid 1980s and early 1990s. (Naur 1985, Lanzara and Mathiassen 1985, Nygaard 1986, Floyd et al. 1989; Floyd 1992, Ehn 1990) As early as 1988, Curtis et al. (1988) published a qualitative interview-based study of the software design process for large systems. Nonetheless, qualitative research on software engineering is infrequently published. Fuggetta (2000) has provided an indication of the state of the art concerning our knowledge of the social side of software engineering in his software engineering process road map celebrating the millennium shift. Qualitative research, and with it social and cooperative aspects of software development, seem to be mainly discussed at workshops. (Sim et al. 2000; John et al. 2005) It was only recently that a special issue of the journal Information and Software Technology aimed to collect qualitative software engineering research from different related discourses (Dittrich et al. 2007, see the editorial for an overview over related research across different discourses).

As an engineering discipline, software engineering is interested in an understanding that also helps to improve the current practice. (Rönnköt et al. 2002) One of the reasons for the still marginal status of qualitative research in software engineering might be that the in-depth understanding of the social and cooperative aspects of software development that qualitative research provides is not easily translatable in the quantifiable causal relationships used to identify and measure both needs for and successes of software process improvement.
In the following section we present CMD and the rationale behind it. We then discuss related research approaches and what distinguishes our approach from these. Section 4 presents four different research projects. The results are discussed in Section 5. Here we also discuss challenges qualitative research provides and how to tackle them. Finally, Section 6 presents the principal results of this article.

2 Cooperative Method Development

Cooperative method development (CMD) takes as a starting point the existing practice of software development in concrete industrial settings. The motivation to develop our own approach was the discontent with how existing research approaches both in software engineering and information systems addressed use-oriented software development. Neither addressed the following questions: ‘How do Software development practitioners tackle their everyday work, especially the cooperation with users around the design of software?’ and ‘How can methods, processes, and tools be improved to address the problems experienced by practitioners?’

A number of researchers in the Scandinavian information systems community have focused on improving the quality-in-use of software by improving software development, and especially the cooperation and communication between users and developers, and within development teams. Researchers participated in projects as designers (see e.g. Ehn 1990), as software engineers developing prototypes (e.g. Christensen et al. 1998), or intensively mediated the use-oriented and participatory development process (e.g. Grønbæck and Mogensen 1997). A whole toolbox of methods, techniques, conceptualisations and tools to improve use-oriented development has been developed, which is well published in the related forums. Critics—often from the software engineering community—doubted whether this toolkit would be applicable, and if it were, wondered which parts would be applicable if researchers had not been intensively active in the projects such as the ones referred to above. This dispute can only be solved by empirical research.

Though motivated by an interest in use-oriented design and development of software, the CMD approach is not specific for these kinds of methods, tools and processes. It is meant as a framework to be further detailed in relation to each specific project. The adaptation can even be limited to the first phase if no improvement is decided on. The remainder of this section presents first the CMD cyclic research model followed by a number of guidelines. These provide a comprehensive theoretical and methodological framework that supports the research but that also allows an adaptation of the framework to the circumstances specific to the project. Table 1 presents an overview over the five guidelines and how it relates to the research approaches we chose for comparison.

2.1 The Basic Action Research Cycle

The CMD research process is modelled as evolutionary cycles consisting of qualitative empirical research, technical and methodological innovation in cooperation with the involved practitioners, and the implementation of these innovations, evaluated through accompanying empirical research. Cooperative Method Development can be seen as a domain-specific adaptation of action research consisting of three phases which can be repeatedly applied in the same context.
Phase 1—Understanding Practice The research begins with qualitative empirical investigations into the problem domain. The empirical research aims at understanding and explaining practices and designs from a practitioner’s point of view. The intention is to understand existing practices based in their historical and situational context, and to identify aspects that are problematic from the involved practitioners’ point of view.

Phase 2—Deliberate Improvements The results of the first phase are then used as an input for the design of possible improvements. This is done in cooperation between researchers and the practitioners involved. The result of this phase is the deliberation of measures that can be expected to improve the situation at hand and that address some of the mutually identified problems.

Phase 3—Implement and Observe Improvements The improvements are implemented. The researchers follow these method improvements as participant observers. The results are evaluated together with the practitioners involved. The result of this evaluation will both summarize concrete results for the companies involved and, for the researchers involved, will build the base for the scientists evaluation of the proposed improvement measures.

Kurt Lewin developed action research in the 1940s and 1950s for the social sciences (Lewin 1947). Action research was introduced as an approach to Information Systems Research mainly via the Tavistock Institute in England, and Enid Mumford (Mumford

Table 1 Comparison between cooperative method development experimentation in software engineering and reflective systems development based on the five guidelines

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Cooperative method development</th>
<th>Experimentation in software engineering</th>
<th>Reflective systems development</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Actions research cycle consisting of 3 phases: Understanding, deliberating change, implementation and evaluation of improvement</td>
<td>Similar: iterative approach to both learn from and improve practice</td>
<td>Action research, but there is no phase model proposed</td>
</tr>
<tr>
<td>2</td>
<td>Ethnomethodological and ethnographical inspired empirical research complemented with other methods if suitable</td>
<td>Quantitative methods can be complemented with qualitative methods to generate hypotheses</td>
<td>Not specified, mainly qualitative</td>
</tr>
<tr>
<td>3</td>
<td>Focusing on shop floor software development practices</td>
<td>Focussing on measurable improvements caused by methods, black-boxing practice and contexts</td>
<td>Not specified, can be Shop floor software development, management of software development, or an SPI group</td>
</tr>
<tr>
<td>4</td>
<td>Taking practitioners perspective when evaluating the empirical research and deliberating improvements</td>
<td>Neutral observer, seemingly favouring management perspective</td>
<td>Not specified, can correspond to the choice regarding the research focus but does not have to</td>
</tr>
<tr>
<td>5</td>
<td>Deliberating improvements together with the involved practitioners</td>
<td>Improvements are based on experiments with new methods</td>
<td>Not specified: can be cooperative deliberation, based on de facto standards, like the CMM, or on management interests</td>
</tr>
</tbody>
</table>

Empir Software Eng
2001). Action research describes a way not only to understand social communities and action, but also to take action in close cooperation with those who are involved, in order to improve the situation. In this way, a deeper understanding of the aspects influencing the situation within the observed community is also acquired. A cyclic process, moving between observation, planning of the action to be taken, and implementation of the action, provides a guideline for how to structure such improvement-oriented research (see Checkland and Holwell 1998) for a comprehensive presentation of the action research cycle. In the CMD approach, the implementation of the improvements and the observation of the resulting practice are collapsed into phase three of the research cycle.

In his presentation of action research in relation to information systems development, Checkland emphasizes the need to further specify the research by defining the area of interest, the concrete research methods to apply, and a theoretical underpinning for the research methods, in addition to framing the area of interest. (Checkland and Holwell 1998, p 13) As in all research, the theoretical underpinning of the research methods and the scoping and theoretical frame applied to the area of interest have to be compatible. Choosing action research as the basic approach, one deals with two areas of concern: the social domain researched—in our case shop floor software development practice (guideline 3)—and the need the changes should address—in our case the problems of the software developers are to be addressed (guideline 4). Guideline 2 motivates the methodological choices regarding the methods applied to understand the area of interest. And guideline 5 constrains the methods how the changes are deliberated. The scopes and frames of the area of interest and the method selections should be compatible. As a result, parts of the motivations below seem to be redundant. The basic constraints the CMD puts on the research are strong enough to ensure this consistency, and flexible enough to provide space to relate e.g. to additional theoretical schools, as long as they are compatible with the five guidelines (Table 1). E.g. the project presented in Section 4.3 focused on interaction design as part of software development and the fieldwork methods were chosen in accordance.

One of the lessons learned from applying CMD was that methods and technical implementation are closely intertwined in software development. In one project, we therefore experimented by combining action research and design research where a design of a technical artifact and its evaluation is part of the research process and argumentation (Nunamaker et al. 1991). That means that both technical innovation and method and process improvement are combined within one CMD cycle. See Section 5.1 for further discussion.

2.2 Ethnomethodological and Ethnographical Inspired Empirical Research

Focusing on software development practices encouraged the use of qualitative methods for the empirical fieldwork, in particular using ethnography and ethnomethodology as a theoretical underpinning. Ethnography uses participant observation as a base for understanding cultures of social settings from a members’ point of view (Harper 2000). Ethnomethodology addresses the methods a social group (an ethne) uses to organise cooperation and communication (Coulon 1995). It is no easy task to reveal knowledge about studied social group’s methods, not even with the help of this theoretical underpinning, but studies steered by a specific hypothesis makes such aims impossible: The use of a hypothesis assumes that you already know what you are interested in. (Rönkkö 2005b: 47–55). Applying this underpinning implies that our research and actions are focused on the everyday work of the software engineers whose practice we observe, and how methods, processes and tools can be improved to address the problems experienced by the practitioners themselves. Hence, we often cooperate with highly motivated software engineers.
In all our projects, we used participant observation to develop a basic understanding of the organisation. The field workers kept research diaries. Project meetings and workshops where methods, tools and techniques were discussed, introduced, or later evaluated were audio taped and transcribed. Often we complemented our field material with semi-structured interviews, document analysis, or additional workshops when a topic interesting for research became visible in the field material. Such interviews and additional workshops were also audio taped and transcribed. Research publications are then based on an analysis in a grounded theory fashion (see Robson 2002 for an introduction). In some cases we applied detailed interaction analysis. Each of the published articles details the fieldwork used and the specific analysis performed. Though we have not applied quantitative methods, the CMD approach does not exclude them if they appear to be useful to complement the qualitative data, e.g. in form of descriptive statistics.

One difficulty for software engineering is that it is concerned with the design, construction and implementation of a hardly visible artefact (see also Brooks 1987 and Rönkkö 2007). That leaves important aspects of the development hard to observe. Section 5.2 discusses our methodological innovation to cope with this situation.

2.3 Focusing on ‘Shop Floor’ Software Development Practices

The motivation to develop the CMD approach was a desire to understand how software developers tackle the everyday challenges of developing useful software. Our area of interest was therefore the work practice of software developers as opposed to e.g. the work practices of management or method departments. Other groups and their practices might be subject to the research as far as they are important to the (aspect of) software development practice that is researched. This way e.g. the users and their practice became part of the research in the project described in Section 4.2.

The term ‘practice’ describes here an established way of doing, e.g. of developing software. This practice is produced and re-produced through the action of those who take part in the practice. In this way, the individual’s action is visible and understandable for his or her peers as meaningful behavior with respect to the common frame of reference a common practice provides.

Practice should thus not be mistaken for ad-hoc behaviour. Instead, it provides a base for ad hoc reactions to situational contingencies, often aimed at reestablishing the normal situation. Suchman (1983) e.g. describes how the history of an invoice that had passed its due date was sorted out, in order to establish a situation where it could be paid in agreement with the practices applied in the observed organization. Rönkkö et al. (2005) present a similar situation when analyzing a steering group meeting working out how to act on delays within a project. The latter case makes visible how a company-wide process model served as a frame of reference for a discussion of how a project in reality deviated from the model, rather than as a blue print for the actual project. Focusing on work practices allows the observation and understanding of how exactly methods, process models and tools become influential in and for software development. This understanding can then be used to improve methods, processes and tools.

2.4 Taking the Practitioners’ Perspective When Evaluating the Empirical Research and Deliberating Improvements

Even when focussing on shop floor software development the perspective under which the observations are evaluated and that guides the choice of the improvements can be one of management or users. The importance of being clear about the perspective applied regarding the improvements was one of the results of earlier experiences of empirical
research (Dittrich 2002): Doing research in cooperation with software developing companies puts researchers in a situation influenced by contradicting interests, hierarchies and personal antagonism. In such situations, research results can have both organizational and personal implications (Suchman 1995; Singer and Vinson 2002). The stance researchers take—consciously or not, open or hidden—will influence what they will be allowed to see.

Through taking a clear stance in cooperating with the software engineers whose practice we observe, our research and action becomes accountable for all parties involved. This stance gives orientation when formulating and following research rules of conduct, such as first checking field material with the people we have observed before reporting to others. In exchange we are given more open access to the practice we are interested in understanding and supporting through method improvements.

We started working with collaborative writing, inviting the practitioners we cooperate with to co-author come of the scientific reports of the research results, as an additional way to assure the quality of our empirical results. This improvement is discussed in Section 5.4.

2.5 Deliberating Improvements Together With the Involved Practitioners

As the perspective applied when deciding on the problems to address, the base on which the change is initiated can differ. The researchers can be the ones deciding on what methods to introduce based on the research state-of-the-art or established standards like e.g. the capability maturity model. We decided on a cooperative approach together with the practitioners we observe. We use our academic knowledge as an input to a cooperative deliberation process. The concrete measures to be tried out are decided together with the involved practitioners, i.e. we cooperate on the improvement, and cooperatively try out and evaluate the chosen improvements. The discussions leading up to the improvements provide in themselves additional input to the research; especially they yield criteria for the later evaluation of the chosen improvements. Like the qualitative research, the cooperative design and planning of improvements helps to actively counter the researchers’ bias.

To moderate the process, researchers can take advantage of a rich set of methods proposed by the participatory design community, addressing the combined design of work and computerized support (see Bødker et al. 2004) for a selection of methods and techniques.

There are both advantages and disadvantages of having the subjects of the research participate so closely in the action taken: In one of our projects the research did not proceed further than the introductory empirical research, since the project partners did not see any need to improve their process. One advantage is that the improvement is well anchored within the team that is to change its work practice. If the change then becomes problematic, other reasons for this have to be explored (see the project described in Section 4.4). Beyond this, it helps in the process of countering the researchers’ bias towards specific methods or specific technical solutions (see projects described in Sections 4.1 and 4.2).

Based on the experiences gained through the projects described below, we complemented the decision about and planning for improvements with what we call ‘qualitative experiments’, which are trial implementations of the proposed improvement e.g. in student projects, that help to evaluate the implications of a proposed change. (Section 5.5)

3 Related Empirical Software Engineering Research Approaches

Even though empirical research on software engineering is a growing field, only few research approaches have combined empirical research with method, tool and process
improvement. To give a comprehensive overview goes beyond the scope of this article. Pointers to relevant qualitative research are provided by (Dittrich et al. 2007). For quantitative empirical research, the Journal for Empirical Software Engineering is the most relevant source. In this section, we discuss two complementary approaches and relate them to CMD; both have been formulated as research approaches combining empirical research and improvement efforts, and both have resulted in a considerable amount of published research. The first is perhaps the most prominent empirical improvement oriented approach in software engineering, i.e. Victor Basili’s experimental approach to software engineering research. By discussing CMD in relation to this approach, the complementary character of a qualitative approach is highlighted. The second approach discussed in more detail is Lars Mathiassen’s reflective systems development (RSD). In relation to Mathiassen’s research approach, CMD can be seen as a more specific frame focused on shop floor software development practice.

3.1 Experimentation in Software Engineering

In his article ‘The role of experimentation in software engineering: past, current, and future’ from 1996, Basili proposes that empirical software engineering research should follow the model of other physical sciences and develop an experimental paradigm (Basili 1996). Software development is understood as an industrial process that can be described by models connecting input variables to the outcome, in terms of measurable productivity and quality. These descriptive models—once established—are turned into prescriptive models and used to plan control future projects. Software engineering researchers need laboratories to test their hypotheses through experiments in controlled environments. As realistic settings are important, research and industry should build up such laboratories together, where researchers can develop and evaluate models that fit the specific company and help to improve the development of software. Industrial practice provides a realistic set of projects for experimentation. The Software Engineering Laboratory at NASA Goddard Space Flight Centre cooperating with researchers from the University of Maryland is designed to be just such a laboratory.

What the proposed research might look like is described in (Basili and Green 1994). As an example the authors report several experiments with key elements of the cleanroom development method: replacing testing by reading and peer review. In the first set of experiments, graduate students and personnel at the Software Engineering Laboratory used reading and testing on programs seeded with errors. The different approaches were quantitatively compared. In a second step, some lab groups in a university course were assigned to use cleanroom development, where unit testing is replaced by peer review, whilst other groups used traditional testing methods. Here again, the different approaches were evaluated regarding their performance in error detection. The results again supported the superiority of reading and peer review versus testing. Two case studies using NASA projects were thereby designed and evaluated. Emphasis is put on the measurable variables making improvement visible in a quantitative manner.

The term ‘experience factory’ (Basili et al. 1994) describes the organisation and tools support data collection from ongoing projects—if possible as quantitative measures—packaging these ‘experiences’ in the form of models regarding dependencies between factors and expected outcomes and the ‘re-use’ of these experiences—that is the application of the models for planning and controlling new projects. New methods can be tested in individual projects and if promising introduced into the whole organisation. Besides being designed as a means for organizational learning, the experience factory can be used as a base for researchers to refine their model of the software development process.
Both CMD and the experience factory are iterative and evolutionary approaches with the goal of simultaneously learning from and mediating the improvement of industrial practices (see guideline 1).

Whilst the CMD approach takes its main methodological inspiration from ethnography and ethnomethodology and can make use of quantitative methods as secondary information source (guideline 2), research in Basili’s tradition regards qualitative research as support for hypothesis generation (see e.g. Seaman and Basili 1997a, b, Seaman 1999).

CMD focuses on how software engineers make methods work in their daily practice, and why and how methods inform practice, rather than the measurable effect their application results in. Both the experimental research approach and the experience factory improvement paradigm black-box exactly the practice which CMD addresses as the central research subject (see guideline 3).

CMD takes the perspective of the shop floor software developers as a starting point (guideline 2.4), whilst the perspective taken by the experience factory related research seems to be that of a neutral observer. As the goal is better control of the development process, the results seem to favour a management perspective.

In the CMD approach, improvements are deliberated together with the practitioners as domain experts (guideline 4). Improvements in the experience factory are based on experiments with new methods.

As Sections 4 and 5 show, the combination of ethnographical and ethnomethodological inspired research, combined with the focus on shop floor software development practices, served as a prism to make visible a number of contextual contingencies and how they influenced the observed practices and the feasibility of method and process improvement. These issues would not be accessible for quantitative research, which requires a prior knowledge of the influencing variable in order to be able to measure them.

3.2 Reflective Systems Development (RSD)

Mathiassen describes the research approach to software engineering that he developed with a group at Ålborg University in Denmark in the following way: ‘The problems, challenges and opportunities involved in systems development practice are considered the starting point for systems development research. Research activities yield experience-based knowledge that leads to new and hopefully improved practices. The knowledge that is developed is both interpretive, that is it helps to understand and make sense of practice, and normative, providing support for performing systems development or improving present practices (Mathiassen 1998, p 80).’ More recent and developed presentations can be found in (Mathiassen 2002) and (Iversen and Mathiassen 2003).

The understanding of systems development practice is based on Schön’s concept of the ‘reflective practitioner’ (Schön 1983). The practitioner is active in research as she/he makes her/his thinking and strategies visible for the researchers and uses the researchers’ results as a support for her own reflection-in-action. ‘[T]he reflective researcher...’, Mathiassen cites Schön, ‘...cannot maintain distance from, much less superiority to, the experience of practice...he must somehow gain an inside view of the experience of practice’ (Mathiassen 1998 p 81).

Research is performed as action research, meaning that researchers take part in industrial software development, software process improvement, and so on. To understand practice, several forms of representations used by the practitioners themselves are reused to capture aspects of the project reality for research purposes. (Mathiassen 1997, chapters 2, 3, 4) Formalizations regarding methods and processes are also understood as helpful representa-
tions (Mathiassen 1997, chapter 5). Where necessary and suitable, action research is complemented by case studies and experiments. (Mathiassen 1998, p 81).

A major research project on SPI in industrial practice can be argued to have implemented his approach. Here, different research groups supported and researched software process improvement efforts in several Danish companies. (Mathiassen et al. 2002).

As already mentioned, RSD was a major inspiration for CMD. However, studying the five guidelines for the CMD approach shows significant differences:

Both approaches are based on action research (guideline 1). However, the CMD approach provides a research process model that allows us to distinguish phases of empirical research aimed at understanding a practice, and the deliberation, implementation, and evaluation of improvements.

RSD did not provide clear guidelines for implementing qualitative research and analysing the field material. We looked for an empirical approach providing more guidance for the fieldwork and the analysis, which was in line with the research focus on shop floor software development practice (guideline 3), and the members’ point of view (guideline 4). We therefore decided on a more structured research process (guideline 1) and on ethnography and ethnomethodology as the major theoretical underpinnings (guideline 2). This guideline resulted in quality criteria for the performed fieldwork, its documentation, its analysis and how to report it in scientific publications.

Reflective systems development takes the practitioners’ perspective as a starting point. However, it is not always decided who the practitioners in question are: the practitioners can be management or the SPI groups as well as the development team. The perspectives of the development teams and their project managers have been taken as a starting point in only one of the subprojects of the major project referred to above. (Iversen et al. 1998; see also Mathiassen et al. 2002) In a recent publication the purpose of a specific project is described as ‘...to enhance managerial usage of software metrics programs...’ (Frederiksen and Mathiassen 2005, p 350). Based on initial experiences with empirical research we decided to be more specific about the perspective we apply. We therefore decided to focus on shop floor software development practice (guideline 3) and take the perspective of these practitioners (guideline 4). This decision provided us with a clear research focus and a clear stance in relation to the organisation we were cooperating with.

In order to retain the member’s perspective from the empirical side of our research in the improvement part, we kept to the guideline of cooperating with the involved software development practitioners regarding the improvements (guideline 5). In the RSD inspired projects this is not always the case. The declared aim of the SPI project was to implement the SEI software process improvement paradigm and to learn about the factors that influence such an implementation. In only one subproject (already mentioned above) was a bottom up approach taken, and reported as ‘problem oriented software process improvement’. Prior SPI efforts had flawed the commitment of the key actors in the organization (Iversen et al. 1998).

The results presented and discussed below show that the decision to focus on shop floor software development practice, which implied other specifications compared to the RSD approach, allowed us not only to refine and complement our methodological approach and tools, (Section 5) but also allowed us to further the understanding of how different contextual aspects affect the deployment of methods, processes and techniques in the respective software development practices we observed (Section 5.2). Thus a more focused research approach led to more focused results.
4 Cooperative Method Development Applied

In this section we present different research projects in which CMD was applied and refined. They address quite different aspects within software engineering; design of flexible and adaptable software; use-oriented development; agile development for e-government applications; and the integration of interaction design and software development. As all the research projects initially aimed at both understanding and improving the observed practices, they all started out by applying CMD. However, depending on the specific circumstances the two projects described in Section 4.3 did not implement a whole cycle. The developers involved where satisfied with their way of working.

Besides evaluating the projects regarding the method lessons learned, the presentations are meant to show the different ways to combine qualitative research with method, technique and process improvement and give an idea of what kind of research results came out of the application of the CMD approach. Each subsection is introduced with an overview of the project research. Then the respective implementation of the CMD approach is described, and finally the method lessons learned are summarised. In Section 5 we discuss the methodological findings across the different projects.

4.1 Designing for Changing Work and Business Practice

The first project addressed with the CMD approach took place in cooperation with the IT unit of a major telecommunication provider and a small software house. The project aimed at exploring techniques to develop a flexible contract handler; a business application keeping the data about certain contracts and computing payments based on these contracts and events thrown by other parts of the software infrastructure. Providing mobile communication is a competitive and rapidly changing business. New types of services are invented and have to be implemented. For the business area the contract handler was developed for this, implied types of contracts based on different events.

With a predecessor of the application, only part of the payments could be handled automatically. The new software should not only be able to handle all known contract types but allow for flexible extension when contract types came about.

A research and development project was set up including three partners; the telecommunication provider contributing the case example; a small software developing company that had developed a flexible database management system allowing for change in the data model during runtime (Diestelkamp and Lundberg 2000); and researchers at the university who were interested in experimentation with techniques to implement different layers of flexibility and how the deployment of techniques changes the relationship between development, maintenance and use of software.

As early as the pre-study phase, the project decided not to use the meta-modelling database provided by the second project partner. The development group nevertheless succeeded in creating a very flexible design solution. The new design provides limited tailoring capabilities through a special administrator interface that is accessible to users with special access rights. It is possible to change the prioritisation of contracts. It is also possible to add new contract types if they work in the same way as an existing contract type. Beyond this, programming is needed, which is supported by a well-designed architecture. The design combines different techniques for implementing flexibility, although it does not deploy more advanced technologies like e.g. the meta-modelling database, reflection or other meta-modelling technologies.
We evaluated the implemented solution against a fictitious system supported by research prototypes we developed as a proof of concept that highly tailorable system was possible. The differences in change effort were surprisingly small. End user tailoring was sometimes traded against maintenance effort (see Lindeberg and Diestelkamp 2001 for a detailed evaluation). Furthermore the implemented design fitted the specificities of the use and the development context and the technical platform and surrounding infrastructure (Dittrich and Lindeberg 2002).

4.1.1 CMD Applied

We began the research by investigating the context and the needs that motivated the telecommunication provider’s IT unit to investigate new technologies for implementing more flexible and adaptable systems. The existing solution was not maintainable and thus could not be evolved to fulfil the needs of the business department (guidelines 3 and 4). We reconstructed the history of the development and change of the software to be replaced by the application under development (Dittrich and Lindeberg 2001). At the same time we started participant observation of the ongoing development project. We took part in and taped meetings during the pre-study (guideline 2). This constituted phase 1 of the CMD cycle, Understanding Practice.

We implemented phase 2, Designing Improvements, by organizing two workshops together with the involved software developers, introducing meta-modelling technologies and designs allowing for adaptation of the software without programming. In parallel, a student project investigated the user interface design for a tailorable contract handler. We introduced the flexible database system and developed a number of prototypes to show the feasibility of a design allowing for the redefinition of contract types without additional programming (guideline 5).

Phase 3, Implement and Observe Improvements, consisted of participation in the project and evaluation of the final design. During the implementation phase of the project one of the researchers involved did some minor programming tasks in order to be closer to the now more intensive technical design and implementation. We finally evaluated the implemented design against a fictitious system based on the research prototypes we developed in parallel (Dittrich and Lindeberg 2002).

4.1.2 Method Lessons Learned

The project described above shows how qualitative research can be combined with improvement—in this case improvement of the implementation technologies deployed by one of the industrial partners. The CMD research process allowed us to confront the latest technologies with the requirements and constraints that the use, development and technical contexts posed on the deployment of the technologies (see also Floyd 1992). The latter would not have been possible to achieve with quantitative methods, as we would not have been aware of influences such as user—developer cooperation on the technical design of the system (see Sections 5.2 and 5.3 for further discussion).

Prototypes played a major role in the research process, as technical innovations were tried out. This motivated the inclusion of the ideas and principles of design research—a structured way of including the design and evaluation of a technical artefact into the research process—in a second research project together with the same telecommunication provider (see also Section 5.1).

The interaction between process and use qualities has been subject to an additional investigation into how the project team managed to implement a process that both accommodated the organizational need for control, and the flexibility to react to emerging user requirements. To investigate this topic we developed a notation to map out not only the
production of different documents and program modules, but also the cooperation and communication between users and developers during the project (see Dittrich and Lindeberg 2004). Figure 1 is included to show a photo of the original map developed during a common workshop and the computer based reconstruction. The joint construction of the map of the project was evaluated in a similar way as group interviews would be evaluated. The map served as a mediating tool during discussions, and visualizes and supports the analysis and evaluation results (see Section 5.2 for further discussion).

4.2 Adaptable Systems Design

CMD based on design research has been applied in a second research project between the above-mentioned telecommunication provider and the university involved. This second project involved the development of a new system to improve the existing communication infrastructure. The project was managed using a cooperative mapping workshop, where the stakeholders, including users, developers, and project managers, worked together to develop a shared understanding of the project requirements. The workshop included a series of meetings and sessions where participants discussed the project goals, identified key features, and agreed on the design of the new system. The results were documented in various forms, including annotated interface prototypes and system test cases. The project was monitored using a milestone plan updated monthly, and reports were provided to the steering and reference groups. The project was initiated in January, with monthly meetings of users, and the final version was released in October. The project was a success, demonstrating the effectiveness of the CMD approach in improving the communication infrastructure.
project also addresses the development of end-user tailorable systems and can be regarded as a second cycle of the project described above. Again, the focus is on the deployment of techniques to implement adaptable systems, and again, prototypes developed by one of the involved researchers played a central role in the research process.

During the first CMD cycle in cooperation with the telecommunication provider, we identified the interaction between different adaptable systems as a bottleneck that put unwanted bounds on the individual system’s flexibility. This defined the focus of the second project. The overall problem addressed was how to make it possible for end-users to tailor interaction between flexible applications in an evolving IT-infrastructure. The study addresses the design and deployment of previously non-existent functionality (Eriksson and Dittrich 2007).

The targeted system was the same one as in the project described in the previous section. To compute payment based on contracts, the system must be supplied with data from other parts of the IT infrastructure. When creating new contract types based on different data this dependency became a bottleneck. The data for such ‘extra payments’ has been handled and computed manually using SQL queries and spreadsheets to assemble the data, which then is exported in ASCII format. This process was conducted once a month for several extra payments, which implied a lot of manual work and potential sources of expensive errors.

The research addressed techniques to help the situation. A prototype was built to explore how a tool could be designed that allowed end-users to manage interaction between flexible applications. Our experiences told us that it is impossible to anticipate the structure of future extra payments or which details will be needed. Therefore, the tool had to be able to communicate with any system in the IT-infrastructure. That way, the addition of a new data source would be possible as well.

4.2.1 CMD Applied

During CMD phase 1, *Understanding Practice*, the initial field studies focused on the work practice of the business department. We visited our industrial partner once or twice a week. During the project described in Section 4.1, we learned that software engineers and users worked tightly together when developing and evolving software. Accordingly we observed and interviewed both users and developers (guidelines 2 and 3). The approach made it possible to observe different aspects that influence the use of different kinds of information systems as well as discover requirements that were unspoken and that the users were unaware of. For example, observing an end user conducting the process of computing an extra payment revealed the need for making manual changes to the collected data. This procedure was not obvious and without the observation it would have been natural to believe that the collected data was correct. These field studies informed the development of the overall research question and also the design of the prototype.

In the beginning of CMD phase 2, *Designing Improvements*, workshops were arranged involving researchers, users and developers, discussing the design of the prototype (guidelines 4 and 5). When implementing the prototype, one of the researchers was stationed at the company 2 or 3 days a week to ensure that the prototype conformed to existing company systems. Field notes together with audio taped interviews and meetings ensured that not only the users’ and developers’ comprehensive comments were remembered but also that minor remarks were incorporated in the design. The prototype was evaluated by three employees involved in the collection of data and the computation of the extra payments, and by one developer involved in the maintenance of the payment.
system. These evaluations were video taped. To allow the users to evaluate the prototype realistically, we reconstructed part of the IT-infrastructure in a local environment and populated it with business data, developing our prototype into a case-based prototype (Blomberg et al. 1996). The researcher performing the evaluation sessions observed, and asked exploratory and open-ended questions to provoke reactions that differed from our expectations.

The developer who worked with maintenance of the regular system evaluated the prototype during a workshop. Advantages were discussed and drawbacks concerning use, tailoring, and expansion of the tailoring capabilities were revealed. Cooperative and individual, as well as technical aspects, became visible. The evaluation highlighted some technical issues that the company could transform into facilities in the existing payment system. The evaluation also resulted in critical issues to be considered in order to achieve and maintain quality in adaptable systems that continuously change. One critical success factor was the necessity of close cooperation between users and developers.

During phase 3, Implement and Observe Improvements, the researcher participated in design meetings concerning the improvement of the payment system. In the meetings, the design of the prototype was discussed and ideas about how to make use of the experiences from the prototype evaluation were discussed. The result was that the developers modified some ideas from the prototype and implemented them in the improved payment system. The tailoring activities shifted from being the users’ work in the prototype to being the work of the developers. The researcher also participated in the project meetings in order to follow the process of how the design of the improved payment system evolved during the project. The researcher participated in the project for approximately 6 months, during the requirements and design phase. A couple of months after the system was put into operation a workshop was arranged where the design of the improved payment system was discussed.

4.2.2 Method Lessons Learned

In order to design and implement research where the design and evaluation of a prototype plays a central role in the research, we complemented the CMD approach with the concept of design research (see also discussion Section 5.1).

Method innovation has been replaced by experiments with prototypes and evaluation of the prototypes. Design research (Nunamaker et al. 1991) served as a complementary paradigm, to design, implement and evaluate the research in this second project. According to design research, the design and development of a (prototypical) information system can be used both to answer technical questions and as a probe to explore requirements posed by the deployment of the technical possibilities, resulting in a study with two integrated perspectives, i.e. a social and a technical perspective. Concerning the social perspective, the developed prototype acted as a case-based prototype (Blomberg et al. 1996) meaning that the prototype was used to explore the users’ work, interaction, opinions and requirements when using the prototype. In this aspect, the prototype targeted the revealing how co-operations issues influenced the deployment of certain technical designs. The evaluation of the prototypes with respect to development processes, the technical environment and use contexts, provided a new insight into the limitations of the technological solution and the interaction between design, development and use.

The analysis of the prototype indicated again that the applicability of techniques to implement flexible systems and how the benefit they provide is evaluated depends on
broader contextual influences. E.g. the interaction between users and developers played an important role (see Section 5.3 for further discussion.)

4.3 Agile Development for E-government

In a third project, we applied the CMD approach when researching the development of two different e-government related systems each developed by a small software provider. The first company developed a booking system for municipalities, with a special emphasis on sports amenities and communication with sports clubs. The second company specialised in web applications facilitating citizen participation, e.g. for city planning. The reason for conducting research in these two companies was to understand the reason for the high level of satisfaction that we found amongst their customers.

Our research focus thus was the communication that took place between different participants in the development practices, and in which way the communication was important for the development of successful software. Both companies had surprisingly similar development practices, despite different technologies, the different kinds of application and the fact that one of them developed and maintained a product whereas the other performed contract development.

- Both companies were small and communication was mostly face-to-face. Common coffee and lunch breaks offered important opportunities for discussions that influenced the projects in a positive manner. Short and quick decision procedures were characteristic for both companies.
- Common for both companies was the frequent and friendly contact with users and/or customers.
- Both companies let user and customer feedback guide future development of their systems. For example, the fundamental business concept employed at one of the companies was to listen to users, and develop software in a way that continues to keep customers and users satisfied.
- Both companies were sensitive to changes in requirements and delivered new updates several times a year to keep the system and web sites up-to-date.
- Employee turnover was low at both companies.
- Both companies were located in small towns. Most of the employees had children and sometimes only worked part-time. Working overtime was very rare. The head of one of the companies stated that their company culture, mixing high social competence with sufficient technical competence, actually constituted a difficulty when hiring new staff.

4.3.1 CMD Applied

As the companies were satisfied with their development process, we did not apply the improvement part of the CMD approach. In relation to CMD this means that phase 2 and 3 were not performed (guideline 1).

By using participant observation, where the researcher participated in the daily routines, it was possible to study undocumented development practices. We could observe how, for example, cooperation, discussions and conversations took place between the different participants in the development projects. Participant observation also made it possible to listen to the conversation between users and developers and the way the communication took place. Interviews gave the possibility to clarify uncertainties and ask about specific
issues in a deeper way. By using multiple ways of collecting data and combining different kinds of methods it is possible to triangulate the finding. We validated our impressions from the fieldwork by constantly referring back to, and crosschecking with, various informants (guideline 2 and 3).

What we observed can be regarded as a user-driven agile software development practice, although the companies did not use the term ‘agile’ when describing their processes (Hansson et al. 2004, 2006). The observations of development practices in these companies cannot be connected to a mainstream software development method or practice by the book. Comparing the development practices of the two companies with the principles and practices of agile development (Cockburn 2002) showed a surprising match.

As the developers were satisfied with their way of working, we did not implement any improvements. With that we adhered to guidelines 4 and 5 with the consequence to restrict the research to understanding the existing practice and to phase 1 of the CMD cycle.

4.3.2 Method Lessons Learned

The qualitative research approach made us focus on social aspects, and also helped us to understand how the work practices were actually conducted. The main findings were not problems connected to the development work, but the identification of good practices. We learned how software developers in practice managed evolutionary development that accommodated late—or even continuous—input of requirements. The main conclusion was the importance of long-term ongoing close cooperation and communication between developers, customers and users to achieve successful development practices.

We started to understand the practices software developers used to develop software with emerging requirements in a way that satisfied their specific kinds of customers. The agile development practices cater to the continuous development of a niche product on the one hand and supported the development of more experimental application in the context of e-government (see Section 5.3 for discussion).

In an interview-based study, we investigated which aspects seem to play a role for companies to choose either an agile or a more traditional approach in a more general way. This study confirmed and refined the factors seen in the two projects described here, but did not provide as rich a picture as the research applied in these two cases (Hansson et al. 2006).

4.4 Integrating Interaction Design and Software Development

In our last case, CMD addressed the development of software for mobile smart devices at UIQ Technology AB. UIQ is an internationally focused company founded in 1999, today with more than 200 employees situated in Sweden. The company develops and licenses a user interface platform for mobile phones using the Symbian OS. The focus of this project was to support the deployment of the results produced by the interaction design group, during the whole development process.

4.4.1 CMD Applied

Phase 1 in our first CMD cycle constituted a 12-month period of ethnographic field studies where the researcher also participated in interaction design meetings, and interviewed interaction designers, a usability researcher, sales and marketing staff, managers and
developers. The aim of this phase was to provide an understanding of the existing practices in their historical and situational context, and to identify problematic aspects (guidelines 2 and 3).

One result of phase 1 was that opinions about what the end users actually want differed between interaction designers and software developers. The Interaction designers often had to confront developers with such questions as—*from whose perspective do you put forward claims? Is your opinion a fair representation of the user’s opinions?* (Interaction designer) The Interaction Design (ID) team wanted to remain faithful to the developers’ creativity and their good intentions while at the same time directing it towards a shared user understanding which ranged beyond their own personal opinions (guideline 4).

This result was the base for the design of possible improvements in phase 2. Here the ‘Personas’ method was introduced by the interaction designers. The method is an interaction design method that helps elaborate and visualize end user categories (Cooper 1999, Pruitt and Grudin 2003). Personas was an especially interesting technique as it was marketed as a remedy for the diversity of opinions regarding design, about who the users might be and what their actual needs and desires are (guideline 5).

It was decided to implement personas in one of the software development projects that simultaneously developed the platform and a specific application for one or more customers. A methods project was started to prepare this implementation. A survey of the Personas method was performed by the researcher, and an academic view of it was presented to the company. The Personas method was also tested in three parallel large team student projects (15–20 students working a 20 week term on the project; Rönkkö 2005a). Discussion seminars were arranged at the company to learn more about Personas in relation to the company’s specific needs. These research and method implementation efforts constituted phase 2 in our first CMD cycle.

Within a time span of one and a half year, and for reasons which were not obvious at the time, Personas were never implemented. Consequently, as the improvement suggestion was never implemented, phase 3 consisted of interviews addressing the organizational reasons for not implementing this improvement. A number of branch related external socio-political contingencies and circumstances make our experienced problems of trying to implement Personas appear reasonable (for more details see Rönkkö et al. 2004).

But what about the communication of the interaction design results? Here we started a second CMD cycle as we expected the original problems would still exist. The *first phase* of the second CMD cycle aimed at understanding the now changed needs of the practitioners involved (guidelines 2 and 3). The new ethnographic results were somewhat surprising, as the initial problem that triggered the need for improvement had disappeared. Developers no longer had to argue strongly for their own opinions or requested changes in interface designs. Also, another developer habit of sometimes rejecting some solutions as ‘not technically feasible to implement’ had disappeared. Our research showed that a number of circumstances had changed since it was decided to implement the Personas method:

- UIQ had decided to *re-organize* due to the increased number of employees. The internal re-organization as such gave the interaction design team recognition for their important role in influencing the company’s existing product. A *new user interface platform* was to be developed. Laying out the design of the new product gave the interaction design team unique knowledge about the product that developers did not possess.
- The *acknowledgement from management*, which was apparent in the above-mentioned task, was influential. With regard to the development of a new platform, the company’s
internal vision of achieving the best UI in the world was re-actualized. Such a vision emphasized by management once again gave the interaction design recognition and credibility.

– Finally, the interaction designers had arranged discussions in which previously suggested and rejected interface ideas were compared to existing designs in the platform. Surprisingly, many of the ideas that developers in earlier discussions had argued were ‘not technically feasible’ were present in the platform. Hence, room for a tougher attitude was indicated. The team decided to act with a tougher attitude when confronting ‘not technically feasible’ responses to design suggestions.

Altogether, these factors placed the interaction designers in a unique knowledge position, and also gave the interaction design team a ‘licence’ from the organization’s side to introduce a more perseverant attitude towards suggested changes of initial design suggestions. Our second CMD cycle ended here (guideline 5. See Rönkkö 2005b, especially pp 237–259 for more details).

4.4.2 Method Lessons Learned

The central results of the project address the interaction between professionally meaningful methods and the organizational and political environment in which they are deployed. First, the Personas method was not applicable because branch related contingencies hindered the implementation. Second, the failure of the method implementation was actually due to external factors. Third, a reinvestigation of the starting preconditions confronted us with a surprise; the needs that the implementation of the Personas method targeted had already been ‘taken care of’ by socio political changes in the organization (for further discussion see Section 5.3).

Ethnography has proven an adequate research instrument to understand the influence of socio-politics on the deployment of methods in software development.

We extended the CMD cycles with qualitative student experiments, to explore the benefits and problems of the proposed method in an isolated context before changing the whole development organization (see Section 5.5. for further discussion).

As a way to combine member checking—a methodological element proposed in ethnography to assure the quality of the analysis—with further clarification, we experimented with joint paper writing. Although the researchers stood for the collection and analysis of the field material and the main part of the text, the industrial co-authors participated in evaluation and at the same time provided further material and insight on the issues discussed. In this way we managed to create a forum for deeper reflection, i.e. rich descriptions were produced which helped us to keep track of subjects studied, even during writing phases, and provided additional insight. This handling of the qualitative results can be perceived as a joint knowledge generating approach (see Section 5.2 for further discussion).

5 Cooperative Method Development Re-visited

The presentation of the four research projects above documents research that took place over a period of 6 years. Each of them applied and appropriated the CMD approach in different ways. They show that and how qualitative social science methods can be combined with the improvement of methods, processes and techniques. Section 4.3
indicates that the CMD approach does afford enough flexibility and sensitivity to also handle a situation when practitioners do not consider their practice as lacking. The improvements that are devised are evaluated in actual development, in order to understand how local contingencies of the actual project, especially the social and coöperational aspects, influence the deployment of methods, processes and techniques.

The research described above has sometimes been a challenging journey. We have learned to cope with an abundance of field materials under non-disclosure agreements, we are still struggling with writing about qualitative projects in a way that is comprehensible for the software engineering research community, and we can only emphasize the cooperation within the research group as the only way to cope with the danger of ‘going native’ (Hastrup 1995). The discussion of the concrete implementation of the research would justify an own article.

In the discussion of this article, we limit ourselves to the lessons learned that relate to the five guidelines of the CMD approach and we will discuss how we complemented CMD based on our experiences. Table 2 gives an overview over how and to what extend the different research projects implemented CMD, and what extensions they suggested. The subsections below correspond to the subsections of Section 2.

As the original version the complemented version is meant as a frame that has to be adjusted and filled out when applied to a new project.

### 5.1 Action Research can be Complemented with Design Research

Basically, the action research cycle consisting of the three phases is confirmed. It provides enough guidance for the cooperation with the software developers involved. Even when there was a decision to not implement any change, the research in phase 1 lead to interesting and publishable results. In the UIQ case (Section 4.4) also the not successful trials for implementation lead to research results as the failure triggered an investigation into its reasons.

When addressing design techniques and methods, our approach benefited from being complemented with design experiments that allow the evaluation of whether certain technologies fit both the intended use context and the development organization (Sections 4.1 and 4.2). When introducing innovative design methodologies, e.g. aiming at the development of tailorable software, the design solution is to be evaluated together with the changes in the development process they might imply. In the first project (Section 4.1) the research prototypes were an isolated activity informing the decision on the design of the program under construction. In the second project, with the telecommunication provider, the design of the prototype served not only as a probe, but became a focus in the research (Section 4.2). We characterised this kind of research by using the notion of ‘design research’ from the IS community (Nunamaker et al. 1991). We not only evaluated prototypes on the basis of their technological qualities, but also used them to discuss implications for the interleaving of development and use in respect to a specific application domain. (Erikson and Dittrich 2007) Different from the action research cycle, the deliberation regarding the new technique and the implementation and observation of its implementation involved the evaluation of a prototype or design.

Design research today is not only discussed as an approach when designing and evaluating a piece of technology as part of the research but is also applied when designing and evaluating methods. From this perspective, the qualitative experiments as discussed in Section 5.5 can be described as design research as well. Even the CMD approach as a whole can be regarded as design research. We decided against this. The emphasis of our research with regard to
<table>
<thead>
<tr>
<th>CMD Guidelines</th>
<th>Telecom proj. 1</th>
<th>Telecom proj. 2</th>
<th>e-Government</th>
<th>UIQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guideline 1 Actions research cycle consisting of 3 phases: Understanding, deliberating change, implementation and evaluation of improvement</td>
<td>Implementation One full CMD cycle</td>
<td>One full CMD cycle. Can be seen as a second cycle to the previous project.</td>
<td>Only phase 1 applied</td>
<td>One full cycle and phase 1 of a second cycle</td>
</tr>
<tr>
<td>Method lessons learned</td>
<td>Confirmation</td>
<td>Confirmation. Complement with elements from design research</td>
<td>Adhering to guidelines 4 and 5 implies that only phase 1 was applied.</td>
<td>Confirmation Adhering to guidelines 4 and 5 implies that only phase 1 was applied in the second cycle</td>
</tr>
<tr>
<td>Guideline 2 Ethnomethodological and ethnographical inspired empirical research complemented with other methods if suitable</td>
<td>Implementation Participant observation, interviews and document analysis</td>
<td>Participant observation and interviews</td>
<td>Participant observation</td>
<td>Participant observation, interviews</td>
</tr>
<tr>
<td>Method lessons learned</td>
<td>Confirmation</td>
<td>Confirmed</td>
<td>Confirmed</td>
<td>Confirmed</td>
</tr>
<tr>
<td>Guideline 3 Focusing on shop floor software development practices</td>
<td>Implementation Development team</td>
<td>Development team and users, as they cooperated closely</td>
<td>Software developers in the respective companies</td>
<td>Interaction design team and cooperating actors</td>
</tr>
<tr>
<td>Method lessons learned</td>
<td>Confirmation</td>
<td>Confirmation</td>
<td>Confirmation</td>
<td>Confirmation</td>
</tr>
<tr>
<td>Guideline 4 Taking practitioners perspective when evaluating the empirical research and deliberating improvements</td>
<td>Implementation Deploying techniques to implement more flexible and evolvable software</td>
<td>Deploying techniques to implement more flexible and evolvable software</td>
<td>The involved developers did not have any problems</td>
<td>Cooperation and communication between interaction designers and other software engineers.</td>
</tr>
<tr>
<td>Method lessons learned</td>
<td>Confirmed</td>
<td>Confirmed</td>
<td>Confirmed</td>
<td>Using co-authoring as a way to establish a common understanding of the problems perceived</td>
</tr>
<tr>
<td>Guideline 5 Deliberating improvements together with the involved practitioners</td>
<td>Implementation Joined workshops, research prototypes</td>
<td>Joined prototype design and evaluation</td>
<td>No changes were devised.</td>
<td>The Personas method was proposed by the involved practitioners</td>
</tr>
<tr>
<td>Method lessons learned</td>
<td>Confirmed</td>
<td>Confirmed</td>
<td>Confirmed</td>
<td>Complementing the deliberation process with qualitative experiments.</td>
</tr>
</tbody>
</table>
methods is not purely the design and evaluation of methods but the understanding of software
development practice and how it influences the deployment of methods.

The design research cycle might replace the action research cycle in a way decided by
the respective project, or it can be implemented in a parallel corresponding cycle—in cases
where the technological innovation and a corresponding change in methods and processes
are subject to research as e.g. in the introduction of a product line architecture approach
described in (Unphon and Dittrich 2008).

5.2 Making Software Development Visible

The ethnographical and ethnomethodological take on the empirical research proved to lead
relevant understanding of the observed practices and a sound base for the improvement
proposals. Even when no change was initiated, relevant research results were achieved.

As part of the research described in Section 4.1, we complemented the qualitative
fieldwork methods with mapping techniques to visualize the complex and often not easily
accessible software development practice. For research purposes, trying to understand
complex networks of formal and informal contacts, meetings, and talks whilst standing near
the coffee machine is extremely difficult. We adapted fieldwork methods from the social
sciences that allow the observed to participate in the representation of their personal
practice (see Dittrich and Lindeberg 2004 for a more detailed description). These methods
have been especially helpful for visualizing the tight cooperation between users and
developers at the telecommunication provider. At the moment, we are experimenting with
similar techniques for other aspects of the software engineering process. Compared with the
use of different kinds of maps discussed by Lanzara and Mathiassen (1985) we use
mapping techniques so far as a complementary field work technique rather than as a means
of intervention.

5.3 Contextual Influence became Visible in the Shop Floor Software Development Practice

The focus on shop floor software development practices provides a prism through which
the influences of organisational, technical and business specific contexts on the deployment
of methods processes and techniques become visible. Thus what might have seemed as a
constraint actually turned out to be an asset.

That qualitative research focusing on software development practice allows the
understanding the communicative and cooperative processes and the influence of contextual
constraints on the method in use is the second finding of this article. It is confirmed in all of
the projects reported above: The suitability of techniques that provide light weight
flexibility can only be argued by taking the business environment, the technical
infrastructure and—last but not least—the close cooperation between the business units
and the software engineers at the telecommunication provider into account (Section 4.1).
The deployment of technologies to allow end-users to tailor the system infrastructure at the
same provider depends on the high level of IT competence within at least part of the user
group, and once again upon the close cooperation between software engineers and users in
the business units (Section 4.2). The research on software development for municipal
applications makes visible the advantages of agile processes when developing for an
evolving domain, especially when addressing more experimental application areas
(Section 4.3). The last project (Section 4.4) proves that qualitative approaches allow
analysis of the contextual dependencies even when the introduction of a method does not
succeed. The failure of the Personas method to communicate the background for interaction
design and usability studies to the other members of development projects resulted in a better understanding of how the organization of a specific branch or business area influences the applicability and adequacy of even very focused methods.

These results can be read as a challenge for software engineering research; the success or failure of an improvement proposal might depend not only on the suitability of the method—it might also depend on socio-political factors that are sometimes only remotely related to the local development practice.

5.4 Taking the Practitioners’ Point of View Countered the Researchers’ Bias

The qualitative research approach in connection with the focus on the practitioners’ problems when deliberating change provided a sound correction of the bias of the researchers towards specific methods and techniques. In the cases described in Section 4.3 and the second CMD cycle of the UIQ case (Section 4.4) this even lead us to restrain from implementing any change.

Qualitative research, especially ethnographic methods, relies on the researcher using him or herself as an ‘instrument’ by observing a community’s reaction to the researcher, and the researcher’s own emotional and cognitive response to the strange community. In particular, it is the differences between the researcher and the researched culture, which become visible in these reactions, which are the basis of the results. (Anderson 1997) This tradition of reflectivity already helps to plan, document and reflect the influence of the researcher on the situation. For example, in both cases of cooperation with the telecommunication provider, (Sections 4.1 and 4.2) the researchers interfered with the situation by exploring and proposing technical solutions for the problems under discussion. The—in many ways unexpected—reaction to the prototypes lead to workshops evaluating and comparing a prototype based fictitious solution and the implemented design and thus contributed to understand aspects of the situation that influenced the applicability of techniques, in ways that we as researchers did not anticipate.

In the two cases presented in Section 4.3, the observed practices—from a software engineering point of view—lacked structure and control. Observing their success regarding user and customer satisfaction kept us from quick improvement proposals and helped us to develop an understanding of the rationale of the observed practice by referring to agile development.

In the UIQ project (Section 4.4) the process of co-authoring research papers provided a framework for agreement on the status of the problems solved, the circumstances that surrounded the solutions, remaining challenges, and new steps to take. That way we also managed to jointly document our experiences without losing important aspects of the social contingencies in the industrial real world.

In all cases the researcher’s influence on the situation is planned, reflected on, and reported in the published results, so that the influence is visible and the distortion of the situation can be evaluated by the research community.

5.5 Complementing the Cooperative Deliberation Process with ‘Qualitative Experiments’

The cooperation with the involved practitioners around the deliberation of methods and techniques to be introduced in their everyday practice turned out to be an additional source for empirical material regarding contextual constraints to method technique and process improvement.

Changing methods and processes, even for project groups of 10 to 20 persons, implies a major risk for every software-developing organisation. The original approach can and should
therefore be complemented with ‘qualitative experiments’ that allow evaluation of the proposed improvements, using qualitative empirical methods in a low risk environment e.g. using student projects. This allows an evaluation of the considered changes without too much loss in case of a failure. Here research, education and cooperation with industry can benefit from each other: When the introduction of the Personas method was delayed, we started a series of experiments using large team educational projects (15–20 students working with the project for 6 months) to evaluate the actual benefits of this method (Section 4.4, Rönkkö 2005a). The result indicated that the goal of improving communication between interaction design and software engineering could also be reached by other means. Such experiments help to understand the contributions of a specific method without interfering with existing development practices, and provide better input for the decision whether or not to take the risk of changing the processes at the industrial cooperation partner.

6 Conclusions and Future Work

Combining qualitative research, a practitioner’s perspective, and a cooperative approach when developing and adapting methods and techniques, proved to be a fruitful approach to understanding the social and cooperative aspects of software engineering, and also to understanding how contextual influences affect the deployment of software development methods, processes, and techniques. What makes a method work—or not—is sometimes not only the qualities of the method or the local circumstances but depends on more remote aspects such as the organization of the whole branch of industry or the relationship to customers and users.

These results would not have been achievable using alternative approaches: Compared to Basili’s experimental approach, which black boxes both the context (as variables) and the concrete practice, the CMD approach helped us to understand which contextual factors influenced the applicability of methods, techniques and tools, and how the observed practice related to the context. Taking the perspective of the involved practitioners as a starting point for the qualitative empirical research, and for the deliberation of improvements, helped us to understand the rationale of software development practices that do not comply with the ideas of the software engineering community of how software should be developed. Ethnography and ethnomethodology provided a suitable methodological frame for the research.

Both reflective systems development (RSD) and CMD combine qualitative empirical research and action research. The distinct cyclic research model and the methodological basis in ethnography and ethnomethodology supported the focus on shop floor software development practices and made our research accountable, both with respect to our research partners and with respect to the scientific community. Although CMD’s focus on shop floor software development practices seems to constrain the research, it actually served in the actual research as a prism to render visible how contextual contingencies influenced the deployment of methods, techniques and processes.

Social science methods have to be adapted and complemented beyond what was anticipated in (Dittrich 2002). The technical design interacts with the development methods and processes. Thus action research has to be complemented with design research. The concrete effort of fieldwork has to take the specific character of software and its development into account. Mapping techniques applied together with the practitioners allow making otherwise hard to observe aspects of software development practices visible. The qualitative experiments allow us to explore the usefulness and the constraints of methods before implementing them in industrial practice.
Empirical research together with industry, and especially action research, often faces the question whether and how it differs from consultancy. Though our research hopefully leads to an improvement of methods, processes, or techniques at the industrial partners, our research cooperation differs from consultancy work in a number of ways: In the projects reported in this article, the industrial partners contributed with the work they put into the projects; the researchers’ contribution was paid by research agencies or by the university. That provided the necessary freedom to spend more time on understanding the existing practice before discussing improvements. The CMD guidelines imply that we take a shop floor development perspective both in the empirical research and in the deliberation of improvements. We do not refer to management in the same way as consultants would. The CMD approach keeps us from ‘selling’ a specific methodology. Changes were deliberated together with the involved developers, and they were implemented by the developers rather than by the researchers. Independently of whether the changes decided on were successful, the research provided a possibility for the involved developers to reflect on their practice and learn about new methodological and technical approaches. Last, but not least, the existing practice, the deliberation process, the changes introduced and the changed practice have to be documented and analysed in an academically vigorous way.

To develop and deepen the understanding of these methodological innovations provides a number of challenges for future research. One is the question of how to learn from the results of qualitative research about other projects within the same organisation or across organisations. For example: We started to experiment with interview based multi-organisational studies in order to understand whether and how certain contextual factors influence different software organisations (Hansson et al. 2006).

In Rönkkö (2007) one of the authors uses sociological concepts to understand the negotiations around a delayed project, that resulted in a specific interpretation of the company’s project model, in order to satisfy organisational accountability with the needs of the project. Such an application of sociological results might result in general explanatory schemes that can be applied more widely.

One additional idea to further improve the CMD approach would be to support the development of communities of practice (Wenger 1998) within or across organisations, which could interact with researchers and serve as a kind of focus group for innovations and improvements. Such an approach would comply with the emphasis of the members’ perspective even when addressing the problem of generalisation.

As this last challenge relates to the well-discussed topic of the generalisability of qualitative research, there will not be any easy solution.

Acknowledgments Thanks to our industrial and public cooperation partners who made the research possible. The Knowledge Foundation in Sweden, under a research grant for the software development project “Blekinge - Engineering Software Qualities” (http://www.bth.se/besq), Rådet för Arbetslivsforskning, and the European Commission sponsored the different projects this article reflects on. The reviewers of this article contributed with their constructive critique to a substantial improvement.

References

Cooper A (1999) The inmates are running the asylum. Macmillan, Indianapolis
Lanza L. G, Mathiassen L (1985) Characterizing situations within a system development project. Information and Management 8, pp. 3–20 (reprinted as Mathiassen 1997, chapter 5)
Rönkkö K (2005a) An empirical study demonstrating how different design constraints, project organization and contexts limited the utility of personas. Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS), January 3–6
Yvonne Dittrich researches and teaches as Associate Professor at the IT University of Copenhagen, Denmark. In 1997, after completing her Ph.D. in Computer Science from the University of Hamburg, Germany, she built up research in cooperation with Industry at Blekinge Institute of Technology in Sweden. Her research combines software development as cooperative work, end-user development, and use oriented design and development of software. Her current work addresses the development and evolution of software products.

Kari Rönkkö is an Assistance Professor in the School of Engineering at Blekinge Institute of Technology. He has 15 years work experience before entering academia. During these years, he gained experiences from areas and branches such as: shipping, nursing, manufacturing industry and vocational rehabilitation. He received the Bachelor and Master degrees in Computer Science with emphasis on Human Work Science 1998 and 1999, and his Ph.D. degree in Software Engineering in 2005 from Blekinge Institute of Technology (BTH). His main research interest is software engineering, especially the social element and usability. He has had substantial interaction with industry within the telecommunications through collaborative research. He is a member of the management team of the Blekinge Engineering Software Qualities – Research Centre, and Manager for the research group Use-Oriented Design and Development at BTH.
Jeanette Eriksson is a Ph.D. student in Software Engineering at Blekinge Institute of Technology in Sweden. Her research area is within tailorable and flexible systems. Her research interests include exploring how to reflect different user aspects in software architectures to achieve software that adapt to altered conditions and requirements in the end user environment. She is also interested in how to involve end users in the technical design process of adaptable software.

Christina Hansson has a Master of Science degree in Computer Science and Human Work Science from Blekinge Institute of Technology, 2002. Her main interests are use-oriented design and design processes.
Olle Lindeberg is a Lecturer at the School of Engineering at Blekinge Institute of Technology. He has a Master degree in Physical Engineering. He was a Software Consultant for 8 years before returning to academia. He took a licentiate degree in Computer Sciences in 1992. His main research interest is software engineering, especially end-user design. He is Head of the Department of Interaction and System Design at Blekinge Institute of Technology.