Challenges of Software Recontextualization: Lessons Learned

Monique Janneck
University of Hamburg
Department of Psychology
Von-Melle-Park 11
20146 Hamburg, Germany
monique.janneck@uni-hamburg.de

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.
ACM 978-1-60558-930-5/10/04.

Abstract
This paper describes the case of a complex and problem-ridden software development and deployment process: The implementation of a Campus Management system at a large university. Based on an understanding of software development as *recontextualization process* on the technical, organizational, human, and task level, critical factors for success or failure are analyzed. Results show that deficits in change management and organizational support account for a considerable amount of difficulties in the implementation process. Furthermore, individual characteristics and commitment of the users involved play a major role. Lessons learned for software introduction processes are discussed.

Keywords
Software Development and Deployment, Recontextualization, Participatory Design, Campus Management Systems, Business Information Systems, Case Study

ACM Classification Keywords
H5.m. Information Systems: Miscellaneous.

General Terms
Human Factors

Introduction

Introducing a new software system in an organizational context is usually a delicate and difficult endeavor. Despite extensive research on software engineering, participatory design as well as technology adoption and appropriation, in practice especially large software projects are prone to be unsuccessful [e.g. 18, 20, 21]. Blowing the budget or time schedule are common troubles. Even more severely, software may fail to meet the requirements of the use context or meet severe user resistance. The reasons for this are manifold: Project planning and management might be deficient or the software quality might simply be bad.

Moreover, a major issue that seems to be neglected even in high-budget, prestigious software projects is *change management*: The introduction of new technology is inevitably tied to organizational and social change, altering work processes and structures [e.g. 22]. Software projects, however, tend to focus predominantly on technology. Managing organizational change as such is costly and difficult. Intertwining technological and organizational developments is even more complex—and also under-researched in terms of success factors and strategies.

This paper describes the case of the development and deployment of a so-called *Campus Management* system at a large European University. The software introduction affected almost all administrative structures and practices and revealed substantial collisions of interest during the course of the project, causing a lot of frustration and threatening the successful implementation of the system.

The software project was analyzed by means of a qualitative interview study in order to identify critical factors for success or failure and derive recommendations for the continuing implementation process.

The paper is structured as follows: In the next section, a view of software introduction as a process of *recontextualizing* formalized action and the difficulties commonly associated with it is presented. Afterwards the research context and methodology is described. The case study is presented in detail in the following sections. Finally, practical lessons learned for software recontextualization and some prospects for further research are discussed.

Software recontextualization

Software development usually starts as *decontextualization* activity: i.e., formalizing human and/or organizational practices and 'translating' them into algorithms computers can execute [19]. The term *recontextualization* refers to a second 'translation' process of bringing these newly formalized and computer-supported activities back into the use context.

The notion of decontextualization versus recontextualization emphasizes that the challenge of software development is not only writing correct code and providing ample functionality, but integrating new technology into its *social and organizational context* (cf. the debate initiated by Dijkstra [8]). Thus, decontextualization is one side of the coin of software development—recontextualization is the other.

However, in practice software development usually focuses on the decontextualization phase. Even software engineering methods emphasizing prototyping, rapid and cyclical development, the involvement of users in the design process, and formative evaluation [e.g. 2, 3, 4, 9, 10] seldom provide precise methods for handling the software introduction process. *Designing for recontextualization* means to address possible recontextualization problems during software development [cf. 11] and to provide a methodical repertoire for introducing new technology within a (work) context, moderating its use and the change of other (work) practices.

Generally, organizational change is often a difficult process as it involves different stakeholders with possibly conflicting interests who fear or actually experience a turn for the worse. However, software projects are specific in some regards, posing unique challenges:

Formalizing human actions for software support often increases *standardization*. While this might help to make work processes more efficient and transparent, it might also decrease flexibility that is needed to cover irregular and unpredicted exceptions. Software developers should investigate the limits of formalization—or *formalization gaps* [cf. 19]—to identify activities that cannot be processed without a high degree of flexibility and should be automated very carefully or not at all.

In addition, formalizing existing practices can upset contexts and actors by shedding light on *informal* (organizational) structures, processes, relations, and hierarchies that were kept in the dark before. Thus, a software development process might uncover already smoldering conflicts, which are consequently attributed to technology as a scapegoat.

Furthermore, new technology also almost inevitably leads to the establishment of *new structures and routines*. As a result, people might have to change their habits and (work) processes or experience a change of position or reputation, with some stakeholders benefiting and others experiencing drawbacks. Moreover, especially with off-the-shelf software, decontextualization and recontextualization do not take place exactly in the same context: The software is implemented for an abstract or idealized use purpose that may vary greatly from its actual use. Therefore, it is often difficult for users to understand the underlying design principles of the software and relate them to their interests and tasks.

Research Context and Methods

The case study investigates the development and deployment of a Campus Management System at a large European university. The software implementation was accompanied by tremendous organizational and technical difficulties, and it was perceived as a heavy burden by many of those involved. Therefore, the goal of the study was to identify factors influencing a successful implementation and to recommend measures for improvement.

Campus Management Systems are typically complex integrated business information systems covering a wide range of administrative and academic processes, such as recruiting, admission, enrollment in study programs and courses, class scheduling, tracking course requirements, exam results, transcripts of

records and so forth. They might even cover career services or alumni relations, fundraising etc. Often they are connected to learning management systems.

In the case study, the new Campus Management System replaced several existing paper-based as well as technology-based administrative systems, standardizing processes to a much greater extent. Basically all employees and students were affected by the software introduction in some way. Many procedures and workflows were completely altered.

Data collection and analysis

For data collection, a total of 35 in-depth qualitative interviews were conducted to grasp the views and experiences of a wide variety of people who were involved the software introduction process at different levels and in different roles. Interview partners included administrative staff, secretaries, lecturers, research staff, deans, project managers, technical support staff, software developers, as well as student representatives.

The interviews (45-90 minutes duration) were audiotaped and transcribed literally according to a fixed set of transcription rules that had been defined beforehand, resulting in about 1000 pages of text. A qualitative content analysis [12, 14] was conducted using a post-hoc approach: A category system was developed inductively from the data and adapted and refined throughout the process of coding. All interviews were coded by three independent raters. A total of about 4200 codings were assigned to specific text passages. About 25% of the coded interview passages were double-coded as relating to "problems".

The category system developed from the data fits a slightly adapted version of Leavitt's [13] well-known *diamond of sociotechnical interplay* (fig. 1), identifying six major categories:

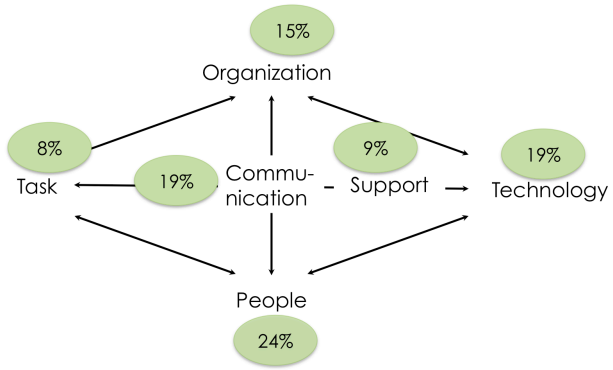

Fig. 1. Adapted Leavitt diamond with percentage of codings in the respective categories.

- "*Technology*" refers to the development process, user participation, and software design (e.g. functionality and usability).
- "*Organization*" refers to organizational structures and processes, such as collision of old and new structures and decision-making powers, and the change management process.
- "*Task*" summarizes sub-categories referring to work routines and roles.
- "*People*" refers to individual characteristics and behavior of the people involved in the process, such as competencies, individual commitment, attitudes, and emotions.

In addition to Leavitt's original model, data analysis revealed two comprehensive categories that were located in the middle of the diamond:

- "*Communication*" relates to information flow, communicational styles, and effects of communication media.
- "*Support*" summarizes all issues related to technical user support services.

A quantitative analysis of codings (fig. 1) revealed "People" as the biggest single category (24%), followed by "Communication" (19%) and "Technology" (19%). "Organization" makes up 15%. The smallest categories are "Task" (8%) and "Support" (9%; miscellaneous: 6%).

About 25% of the coded interview passages were additionally coded as relating to "problems". The occurrence of "problems" in the single categories mostly resembles the original distribution, with one exception: Only 15% of overall codings, but 26% of problems mentioned are allotted to "Organization"—apparently a great deal of difficulties in the software development process is due to organizational issues (table 1).

Table 1. Comparison of overall distribution of codings and problem distribution.

Category	% of overall codings	% of problems
Technology	19%	19%
Organization	15%	26%
Task	8%	7%
People	24%	23%
Communication	19%	16%
Support	9%	7%
Misc.	2%	6%

In the following sections, the results of the case analysis are described in detail. Combining the view of software introduction as recontextualization process with the Leavitt diamond, the analysis is structured along the categories of *Technical Recontextualization*, *Organizational Recontextualization*, *Task Recontextualization*, and *Human Recontextualization*. The diamond is also used to graphically illustrate the findings.

Technical Recontextualization

The development process is characterized by a rather long period of product choice (about 20 months) followed by a rather short process of requirements engineering and implementation until the software was launched (about 9 months).

This tight schedule was due to the adoption of new Bachelor and Master study programs, which were to be administered mainly with the new software. The tight schedule put enormous pressure on the project. Furthermore, many people felt that the project management was incapable and unprofessional.

The software finally chosen among those examined existed in an early beta version and was to be adapted continually over several years to meet the specific needs of the university. This long-term tailoring process

offered the chance for a truly *participatory design* process. Unfortunately, however, the opportunities for involving users in the process were hardly seized. Despite a requirements analysis conducted by employees of the university administration, the vast majority of respondents said that they had not been questioned about their particular work practices, requirements, and needs.

Rather, user participation in the development process was strongly dependent on users' willingness and abilities to *involve themselves* in the process and put their views to the table instead of waiting to be asked by the software developers.

Accordingly, two different categories of requirements engineering measures were identified:

- *Pull actions* were taken by the users themselves to gain influence on the development process. Not surprisingly, only a small minority (n=5) reported taking such actions (e.g. sending unsolicited feature requests, inviting the development team to their department, offering help). These few active users were able to gain considerable influence within the software development team, along with significant decision-making powers (one interviewee called this her "co-developer's privilege").
- On the other hand, the software project team initiated *push actions* to involve users. About one third of the respondents reported that they had experienced such activities (e.g. interviews, surveys, inquiries related to work flows).

Furthermore, the requirements investigations carried out by the project team were mostly on a rather abstract level, making it difficult for the respondents to establish relations to their everyday work practices. For example, supervisors were asked to match the roles implemented in the software to their current staff without truly understanding the functions of these roles. (As one interviewee put it: "I was asked to make a salad without knowing the ingredients"). Low-threshold, user-activating methods like scenarios [5, 6] were not used.

Many respondents said that user acceptance of the software implementation and the changes that came along with it would have been increased if more users had been involved in the process. Since the Campus Management system was so unpopular or even outright rejected by its users it became an easy scapegoat for all problems somewhat related to it, even if they were rooted in organizational difficulties (see next section).

Furthermore, many users felt uneasy with the new software because far-reaching administrative decisions suddenly seemed to be made by a technical system rather than human experts. For example, students believed that admission decisions were automated, which was not the case. Nevertheless, users felt to be at the mercy of a powerful yet intransparent machine.

To sum up, technical recontextualization was flawed by inadequate user participation. Push measures of requirements engineering were methodically inadequate and reached only a small number of people. Pull measures taken by a few, but very active users dominated the process, leading developers to the

misconception that they were in good contact with users (fig. 2).

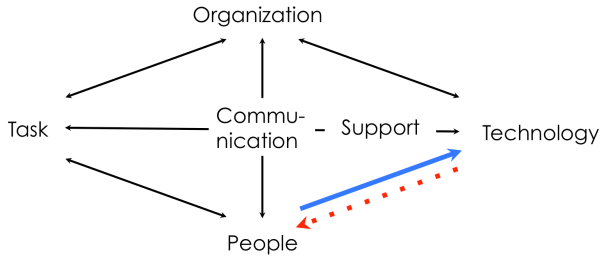

Fig. 2. User participation was driven by a few very active individuals, excluding the vast majority of users.

Organizational recontextualization

The software introduction was linked with far-reaching structural changes within the university. On the one hand, originally independent departments were rearranged and combined into greater units. On the other hand, the introduction of new Bachelor and Master study programs led to new challenges and structures. Furthermore, the university executive board wished to centralize and standardize administrative processes throughout the university, which was to be supported by the Campus Management software.

However, these structural changes were not without conflict. "Collisions of old and new structures" form the biggest subcategory in this area, raised by two thirds of the respondents. The difficulties resulting from these reorganization processes were partly independent from the software implementation: Quite typically, organizational change in large and heterogeneous organizations is difficult and slow. (One interviewee metaphorically called the university a "supertanker", changing track slowly and hesitantly. Quite similarly, another respondent spoke of the software introduction as an "iceberg" colliding with the "Titanic", causing quite some damage).

Many respondents expressed their understanding or even approval that administrative and academic processes and structures were to be simplified and streamlined to a certain extent. However, they harshly criticized that the changes they experienced were shaped rather by the existing functionality and technical possibilities of the software than justified from an organizational point of view.

Again, most people felt that they did not have ample opportunities to participate in the reorganization process—either because they had not been asked or because they lacked experience in change processes. The latter found it difficult to foresee the consequences that certain aspects of formalization (e.g. assignment of roles to certain people) would have. Therefore, even those in charge were often reluctant to make decisions.

It is interesting to note that quite contrary to the personal experience of many respondents, a significant number of changes requested by users were actually implemented in the software. However, as described above, only a small group of especially active users were able to contribute in this way.

Furthermore, the implementation of very specialized requests of selected user groups resulted in an increase of overall software complexity and a decrease of actual standardization, which had been one of the explicit goals of the software introduction. Nevertheless, most

organizational units felt restricted and forced to give up their well-working routines to the software.

It is interesting to mention that the technical support service was evaluated very positively. Since the support staff was extremely knowledgeable and dedicated they were even able to help with organizational issues, buffering the frustration and anger many users experienced to a certain extent. A counterpart delivering "organizational support services" would probably have had a very positive influence on the process.

In summary, organizational recontextualization suffered from a lack of user influence on the reorganization process. As a result, users felt that technology shaped their organizational processes and structures, and not the other way around. There was no explicit change management to accompany the software introduction, even though some difficulties were alleviated by the very dedicated and qualified technical support service (fig. 3).

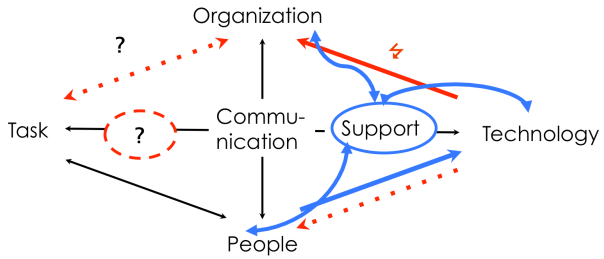

Fig. 3. Technology shapes organization, while users experience a lack of influence on the reorganization process and a lack of organizational support.

Task recontextualization

A central problem related to work (re-) organization was the implementation of access rights and user roles within the software. Originally, for reasons of data protection, the access hierarchy was extremely rigid, colliding with far more flexible and often overlapping real-world practices and roles. Some actors, e.g. secretaries, were not modeled in the software at all, even though they handled central tasks supported by the Campus Management system. (For example, professors were supposed to enter their students' grades into the system themselves, while in reality this was usually done by their secretaries).

Therefore, many work routines had to be altered considerably—or they conflicted with the Campus Management system. Furthermore, some actors experienced forms of debasement because some of their tasks were taken away from them or automatized altogether. Understandably, those persons were quite frustrated—and some of their competencies and know-how were lost for the organization. Again, people were especially frustrated because they felt that they had only marginal possibilities to get involved in the process. Quite alarming from a work psychological point of view, the interviewees reported basically no attempts to influence their work (re-) organization. Quite a few of them showed signs of weariness and fatigue. (One interviewee said: "We're only little wheels in a big machine, spinning round and round").

The Campus Management software plays a tremendous role in many employees' everyday work. By now, many of them spend several hours a day working with the system. Regarding work (re-) organization, many respondents felt that tasks became more complex and

time-consuming. Learning to use the complex software system was difficult for many employees. Usability problems and missing functionality further contributed to these troubles. Thus, more than a year after the software was introduced many employees still experienced it as a burden rather than making their work easier.

To sum up, again technology was experienced to shape work practices, rather than the other way around (fig. 4). Especially the implementation of user roles and access rights in the software turned out to be a critical point.

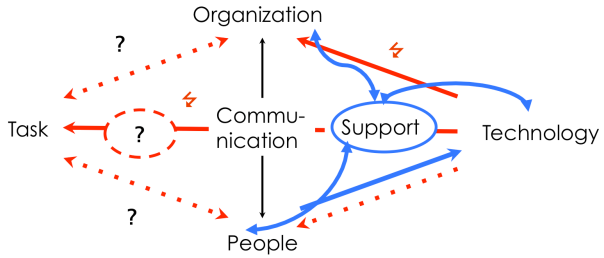

Fig. 4. Technology shapes work, while users have little influence on the reorganization of their work tasks.

Human Recontextualization

As mentioned before, human factors (the "People" category) make up the largest proportion of codings, because the software introduction was accompanied by many emotions and description of the respondents' personal situations took up much room. Furthermore, the analysis shows that individual strategies, commitment, hardiness, and determinedness to participate were essential for individual involvement in the software development process and its respective impact.

The tight schedule of the implementation process and the partially premature and inadequate status of the software put a considerable burden on the staff. Many employees did and do work extremely long hours. Other tasks were often neglected, leading to problems in the respective fields.

Therefore, the manner of the software introduction, especially the timeline, was harshly criticized. While none of the respondents doubted the usefulness of the Campus Management software as such, the modalities of its introduction bred frustration and impaired acceptance. The respondents felt exhausted and exploited and complained about a lack of appreciation for their hard work and accomplishments: They felt that the university administration took all their efforts for granted.

Communication patterns further contributed to this picture (fig. 5).

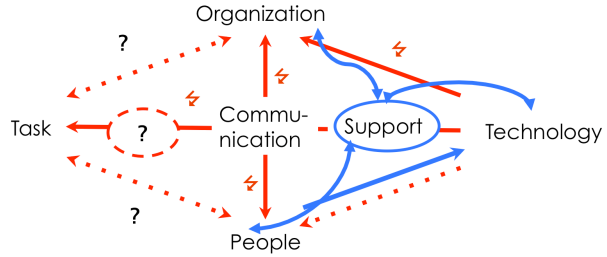

Fig. 5. Communication patterns led to conflicts.

The flow of information was uneven, and too often important information did not reach the intended

recipients, who were frustrated and felt left out. Furthermore, information was often spread informally. Consequently, people who had a tight personal network were better informed.

As described above, personal commitment and the use of pull strategies was crucial for involvement in the requirements engineering and software development process. Strategies named by the respondents include stubborn inquiries, self-initiated contacts to the development team, the use of informal contacts and networks, unsolicited bug reports and feature requests. Quite generally, those who managed to get involved studied the software development and organizational change processes very intensively and became key contacts for the development team, which relied on their judgments and expertise. These key persons were able to push and enforce quite a few special requests for their respective departments or institutions.

Furthermore, personal attributes, competencies, and preferences, such as time and resource management, problem solving strategies and so forth were found to affect individual participation and influence regarding the software development process.

Discussion and Lessons Learned

Like in many software development and deployment processes, tremendous difficulties and resistance accompanied the software recontextualization discussed here. Problems were due to a overly tight schedule, shortcomings regarding the requirements engineering process and user participation, organizational difficulties such as deficits of change management, a mainly technology-driven reorganization process as well as deficits in communication. Quite interestingly, purely "technical" and usability problems (as a subset of the "Technology" category) play a minor role in the interview reports, even though the software still had many weaknesses.

The results presented here stem from a unique context. Universities are surely different from other organizations in terms of management structures and staff. Still, the results as such reflect issues and challenges highly generalizable to other contexts, such as participation, change management, and communication. In this sense, Campus Management Systems can be seen as instances of large business information systems, and implications drawn from this context will most probably be relevant for other software recontextualization processes as well.

In the following paragraphs, practical lessons learned as well as implications for further research are discussed.

Implications for Practice

The case study strongly emphasizes the need for *joint technical and organizational developments*: A tremendous amount of difficulties can be traced to deficits in change management, far more than to technical problems (cf. table 1). However, resources in such projects are typically allocated reversely: Much more money is usually spent on software than on organizational development.

Likewise, the technical user support service was evaluated quite positively in our analysis. The support staff was well-equipped and extremely competent and knowledgeable, while organizational change management had to be shouldered by the respective

organizational units themselves, which were understaffed and under-qualified for the task. Therefore, large organizations should consider installing an *organizational support team* equivalent to technical support structures when introducing new software. Of course that means that ample resources need to be allocated here.

Furthermore, the case study underlines the importance of *user participation* in software development projects, as it is emphasized by the Participatory Design (PD) tradition [e.g. 1, 17]. The case analysis also highlights an issue that is less discussed in this research area: The *selection*—or *self-selection*—of those to involve in the process. The case study shows that barriers to individual participation need to be kept very low, and that heterogeneous user groups need to be involved. Approaches from *Distributed Participatory Design* [e.g. 7] dealing with new methods of user participation in highly dispersed, volatile, and heterogeneous user communities might be useful in this regard.

Pull strategies in requirements engineering and support calling for users' own initiative should be abandoned in favor of *push strategies* of developers actively approaching users. Furthermore, such activities need to be low-threshold and relate directly to users' everyday experience. They need to be thoroughly accompanied by skilled moderators: In the case study, the administrative staff carrying out the requirements analysis had no former experience with the task.

Furthermore, *communication* processes are crucial in two ways: One the one hand, top-down communication (i.e. from the University executive board or project managers to the staff) can be used to give employees a sense of accomplishment and appraisal. In the case study many interviewees voiced their disappointment that they had never received a sign of appreciation. Furthermore, a transparent and honest information policy about what to expect from the software introduction might appeal to people's sense of fair play and increase acceptance.

On the other hand, when communication is organized deficiently, important information might simply not reach the recipients in time or at all, especially when it needs to travel over long organizational distances. To ensure a vital flow of information (especially in large organizations) *communication knots and junctions* need to be defined. That means to establish responsibilities for transferring information to certain people or units according to a fixed set of rules, including feedback loops to make sure the information actually reached the intended recipient. The case analysis shows: The less communication paths were planned beforehand, the more communications chains were likely to break—often without senders' notice. Actions to take could be as easy as setting up correct mailing lists for specific groups.

Furthermore, senders should think carefully about when to use what communication media. In the case study, for example, many people preferred personal contact over e-mail for certain requests. However, most users were offered only standard web forms for inquiries. A lot of information was posted on homepages or sent via mass mailings: Again, this shows a tendency to utilize *pull* rather than *push* strategies, which would establish more active communication with users.

Last but not least, the *software design* itself suffered from several shortcomings. Like many other business information systems, Campus Management systems are very complex. Regarding usability as well as flexibility of the system it might not be advisable to formalize and model the totality of processes in the application domain.

In the case study, trying to model a wide variety of particularities and exceptional cases led to an overwhelming complexity and finally compromised the overall operational concept and consistency of the software. Considering the heterogeneity of requirements and workflows that our analysis revealed in a single organization, a flexible system allowing users to incorporate alternative ways of doing things might be more appropriate and efficient.

In some cases this might be reached by individualization or tailoring options.

Nevertheless software developers should also explicitly strive to identify *formalization gaps*, i.e. processes or structures that should not be modeled in the system at all or to a very small extent. Of course, this is deeply intertwined with organizational developments: e.g., the university administration might have to accept a lower level of standardization.

The Leavitt diamond of recontextualization on different levels proved to very helpful as an *analytical tool*: Using this model, critical factors in recontextualization processes can be evaluated and illustrated in an easy-to-understand and descriptive way. In the case study, the Leavitt diamond showed a strong *predominance of technology* with software-driven one-way interactions, while the goal should be to strive for a more balanced diamond.

Table 2 summarizes main findings of the case study and the implications that can be drawn from them.

Table 2. Summary of main findings and lessons learned.

Category	Summary of main findings	Critical issues and implications
Technical Recontextualization	▪ Unrealistic project timeline ▪ Insufficient user participation ▪ Push measures of requirements engineering were methodically inadequate and reached only a small number of people ▪ Pull measures taken by a few, but very active users dominated the process ▪ Very complex, hard-to-use software design	▪ Use push strategies for user participation ▪ Employ low-threshold, user-activating methods of requirements engineering (such as scenarios, mock-ups, socio-technical walkthroughs, user workshops etc.) ▪ Strive to involve heterogeneous (especially passive) user groups ▪ Identify formalization gaps to avoid overly complex or rigid software designs
Organizational Recontextualization	▪ Technology-driven rather than requirements-driven organizational development process	▪ Strive for joint technical and organizational development

Category	Summary of main findings	Critical issues and implications
	▪ Lack of employees' influence on the reorganization process ▪ No explicit change management to accompany the software introduction ▪ Almost no resources for organizational development	▪ Establish organizational support service as counterpart to technical support service ▪ Allocate ample resources to change management ▪ Allow and encourage employee participation in change management process
Task Recontextualization	▪ Technology-driven rather than requirements-driven reorganization of work processes and tasks ▪ Implementation of user roles and access rights did not match real-world conditions ▪ Almost no attempts of employees to influence their work (re-) organization	▪ Avoid purely technology-driven changes to work organization ▪ Avoid overly complex and rigid implementations of access rights and roles ▪ Allow and encourage employee participation in (re-) organization of work processes and tasks
Human Recontextualization	▪ Individual attributes, competencies, and commitment were crucial for involvement and decision-making power of users in the software development process ▪ Employees felt overstrained and neglected, resulting in further rejection of the software deployment	▪ Account for individual characteristics of users or user groups (e.g. different levels of computer skills, attitudes) ▪ Acknowledge and possibly compensate for employees' additional work load
Communication	▪ Uneven flow of information ▪ Important information often did not reach the intended recipients ▪ Predominant use of push media, such as mass mailings ▪ Information was often spread informally	▪ Define communication knots and junctions as well as communicational responsibilities ▪ Carefully choose communication media (e.g. mass vs. personal communication) ▪ Balance pull and push communication ▪ Try to integrate informal communication
Support	▪ Highly dedicated and qualified technical support service was able to alleviate organizational difficulties ▪ Users showing proactive behavior benefited more from support services	▪ Use push strategies rather than pull strategies in user support

Implications for Research

The importance of joint technical and organizational developments is emphasized by many researchers [e.g. 22]—albeit disregarded in practice. Therefore, more research is needed on how to actually transfer these findings into practice, and how to support software developers and organizations in implementing projects of joint technical and organizational developments. The perspective of *software introduction as recontextualization* proved useful to shift focus from software characteristics to organizational and human factors. Ideally, recontextualization issues should already be taken up in the *decontextualization* phase of modeling, e.g. by devising measures of how to address challenges of recontextualization in early phases of requirements engineering [11].

The *individual characteristics* of users influencing software development processes identified in this study—e.g. personal strategies and competencies or relationships and networks within the organization—have been less researched so far. This is even true for research on technology appropriation, which focuses mainly on the different roles people impersonate, such as mediators, or technological champions [15, 16]. Our analysis shows that individual characteristics influence technology acceptance as well as the degree of participation in the development process: Participating users were largely self-selected. Therefore research should focus more on users' individual characteristics, e.g. regarding personality traits.

As was mentioned above, methods from *Distributed Participatory Design* (DPD) [e.g. 7] might prove useful to achieve user participation in a large and heterogeneous organization. Vice versa, results from the case study might inform DPD research. While DPD is investigated typically in obviously distributed settings like online communities or virtual networks, the case study points to another, less obvious form of distribution: A seemingly homogeneous organizational setting with stakeholders whose diverse interests might only emerge over time and pose a challenge for participatory design. Such forms of *intraorganizational* distribution should be subject to further research.

References

[1] Asaro, P. M. Transforming society by transforming technology: The science and politics of participatory design. *Accounting, Management and Information Technologies, 10*, 4 (2000), 257-290.

[2] Boehm, B. W. A Spiral Model of Software Development and Enhancement. *IEEE Computer, 21*, 5 (1988), 61-72.

[3] Beck, K. and Andres, C. *Extreme Programming Explained: Embrace Change*. Second Edition, Addison-Wesley, 2004.

[4] Beck, K. and Fowler, M. *Planning Extreme Programming*. Addison-Wesley, 2000.

[5] Bødker, S. Scenarios in user-centred design — setting the stage for reflection and action. *Interacting with Computers*, 13, 1 (2000), 61-75.

[6] Carroll, J. M., Rosson, M. B., Chin, G. and Koenemann, J. Requirements Development in Scenario-Based Design. *IEEE Trans. Softw. Eng. 24*, 12 (1998), 1156-1170.

[7] Danielson Oberg, K., Gumm, D., Nagsh, A. (eds). Special Issue on Distributed Participatory Design. *Scandinavian Journal of Information Systems, 21*, 1 (2009).

[8] Denning, P. J. A debate on teaching computing science. *Communications of the ACM, 32*, 12 (1989), 1397-1414.

[9] Floyd, C. and Gryczan, G. STEPS - a Methodological Framework for Cooperative Software Development with Users. Proc. East - West International Conference on Human Computer Interaction (1992).

[10] Floyd, C., Mehl, W.-M., Reisin, F.-M., Schmidt, G. and Wolf, G. Out of Scandinavia: Alternative Approaches to Software Design and System Development. *Human-Computer Interaction, 4*, 4 (1989), 253-350.

[11] Gumm, D. and Janneck, M. Requirements Engineering for Software Recontextualization. *Proc. IRIS 30*, Department of Computer Sciences, University of Tampere, Finland (2007).

[12] Kvale, S. *InterViews: An introduction to qualitative research interviewing*. Sage, Thousand Oaks, 1996.

[13] Leavitt, H.J. Applied organizational change in industry: structural, technological and humanistic approaches. In March, J. G. (ed), *Handbook of organizations*. Chicago: Rand McNally, 1965.

[14] Miles, M. B. and A. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook*. Sage, Thousand Oaks, 1994.

[15] Orlikowski, W.J., Yates, J., Okamura, K. and Fujimoto, M. Shaping Electronic Communication: The Metastructuring of Technology in the Context of Use. Orga*nization Science, 6*, 4 (1995), 423-444.

[16] Scheepers, R. Key role players in the initiation and implementation of intranet technology. *Proc. IFIP WG 8.2*. Boston: Kluwer (1999), 175-195.

[17] Schuler, D. and Namioka, A. *Participatory Design: Perspectives of Systems Design*. Lawrence Erlbaum Assoc, 1993.

[18] Shenhar, A.J., Tishler, A., Dvir, D., Lipovetsky, S. and Lechler, T. Refining the search for project success factors: a multivariate, typological approach. *R&D Management 32*, 2 (2002), 111-126.

[19] Simon, E., Janneck, M. and Gumm, D. Understanding socio-technical change: Towards a multidisciplinary approach. In J. Berleur, M. I. Nurminen & J. Impagliazzo (Eds.), *Social Informatics: An Information Society for all? In remembrance of Rob Kling*. Boston: Springer (2006), 469-479.

[20] Standish Group. *Extreme CHAOS*. Standish Group, USA, 2001.

[21] Torp, O., Austeng, K. and Mengesha, W.J. Critical Success Factors for Project Performance: A Study from Front-End Assessments of Large Public Projects in Norway. *Proc. NORDNET*, 2004.

[22] Wulf, V. and Rohde, M. Towards an Integrated Organization and Technology Development. *Proc of the Symposium on Designing Interactive Systems*, New York (1995), 55-64.