Computer Supported Cooperative Work (2011) 20:403–448
DOI 10.1007/s10606-011-9148-9

Supporting the Collaborative Appropriation of an Open Software Ecosystem

Sebastian Draxler & Gunnar Stevens
University of Siegen, Hölderlinstrasse 3 57068 Siegen, Germany (E-mail: sebastian.draxler@uni-siegen.de; E-mail: gunnar.stevens@uni-siegen.de)

Abstract. Since the beginning of CSCW there was an intense interest for research on workplace design using tailorable applications and sharing customizations. However, in the meantime the forms of production, distribution, configuration and appropriation of software have changed fundamentally. In order to reflect these developments, we enlarge the topic of discussion beyond customizing single applications, but focusing on how people design their workplaces making use of software ecosystems. We contribute to understand the new phenomenon from within the users' local context. By empirically studying the Eclipse software ecosystem and its appropriation, we show the improved flexibility users achieve at designing their workplaces. Further the uncovered practices demonstrate, why design strategies like mass-customization are a bad guiding principle as they just focus on the individual user. In contrast we outline an alternative design methodology based on existing CSCW approaches, but also envision where the workplace design in the age of software ecosystems has to go beyond.

Key words: appropriation, CSCW, end user development, software eco-systems, tailoring, workplace design, eclipse

1. Introduction

> *"How do users design their digital workplaces in an age of open, dynamically evolving software ecosystems?"*—this presents the overall guiding question in this work.

In the past, the question of tailoring working environments was intensively investigated by HCI as well as CSCW research (i.a.Mackay 1990; MacLean et al. 1990; Mørch 1997; Kahler 2001; Lieberman et al. 2006). This is no wonder as inappropriate designed workplaces are considered to interfere with work instead of supporting it. Searching for reasons why supporting workplace design is so complicated, Henderson and Kyng (1991) argued that the worlds' complexity itself makes it difficult for the designer to anticipate all that will eventually be of importance in the users actual work situation. And even the best design can not solve the problem once and for all, if it is inflexible. The strategy therefore must be to design customizable applications that can be adapted within the use

situation in order to fit to the personal preferences as well as to the task at hand. Early solutions did focus on the individual user only, but empirical studies conducted in the CSCW community made aware of this issues' collaborative dimension.

Standing on the shoulder of this research, this work explores a paradigm change in software development with regard to possible futures of the workplace design at the shop floor. This paradigm change can be categorized by several trends like the establishment of the Internet as the dominating infrastructure for mass communication as well as the dissemination and marketing of digital goods; the establishment of new business- and development-models that foster a gift culture, encouraging users to share software with others; the establishment of loosely coupled networks of manufacturers, semi-professionals, and hobbyists, creating small-scale components which can be individually assembled by users.

Messerschmitt and Szyperski (2005) coined the term "software ecosystem" to label this new paradigm. It is semantically related to concepts such as production networks or network economies; however, it tries to integrate the economical and the technological point of view. A Software ecosystem can be defined as a network of related actors, interacting with a shared market (Vasilis et al. 2009). These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts. Technologically, software ecosystems need new architectures to integrate assemblies of coexisting and coevolving software in a deep and seamless manner, to be perceived by the user as a unit solving the task at hand (Bosch 2010).

This new paradigm has been anticipated to some extent by McIlroy's (1968) vision of component-based software development. As early as 1968, he saw future application development as a plugging together of different components bought on the free market. He envisaged the role of a general contractor, offering application services similar to roles in the manufacturing industry; yet today this task often becomes the responsibility of end users. Like many things in life the new responsibility has a twofold character: while it introduces a new freedom to create personal software portfolios, it also requires new competencies to keep an overview over useful and trustworthy material available on the software ecosystem and competencies to assemble them in a reliable way.

The discovery of the local appropriation practices and the change in the global software development practices give reasons to take a closer look on the new paradigm and how this is reflected in the users appropriation practices. Yet, there is still a lack on empirical research on the situated practices, people employ to manage and share personal software portfolios while coping with the complexity of dynamically evolving software ecosystems. This work therefore attempts to address this gap by studying Eclipse as one of the most vivid software ecosystems today. Eclipse is based on an advanced software architecture where 'everything is a plug-in' (Gamma and Beck 2003). Build on this architecture

more than thousand plugins are available on the Internet, provided by a large number of open source projects and commercial vendors.

Our research agenda for investigating into workplace design in the age of software ecosystems is constituted by the following questions:

1. How is the coexistence and coevolution of software structured at the large scale of software ecosystems?
2. Is there evidence that people make use of the new opportunities (including the integration of coevolving software pieces—a work that was previously done by designers)?
3. How is appropriation structured and what situations contextualize this work?

Contributing to the conceptual foundation of CSCW, we further use the empirical findings as a sensitizing lens to adapt existing approaches to support the appropriation work (Pipek 2005) in the local context. Keeping the advantages of software ecosystems, but lowering the burden of using of it, we especially address the following issues:

A. How can we support individual persons when selecting appropriate tools from the ecosystems to design their workplaces?
B. How can we support collaborative appropriation practices?
C. How to foster the collaboration among the ecosystem in mediating the local global context a better way?

Guided by our research topic this work is structured as follows: Section 2 gives an introduction into the related research on workplace design and appropriation work. Section 3 outlines the mixed method research approach we applied, discussing the relation between the ecosystem analysis, the survey and ethnographically oriented study in detail. Section 4 gives a brief introduction into the Eclipse ecosystem, illustrating, how the Eclipse ecosystem functions as a decentralized, open production network. This addresses our first research question. Against this backdrop, Section 5 presents the findings of the survey, contributing to the second research question. Section 6 presents the findings of the in depth case study, answering the third question. Section 7 interprets the empirical findings in terms of technological and organizational opportunities to support the appropriation of software ecosystems. In Section 8 we give a conclusion and we discuss the transferability of our findings in more detail.

2. Workplace design as “artful integration”

In this section we want to draw a rich picture of designing workplace at the shop floor. The tour is guided by our primary research interest: workplace design in the age of software ecosystems. Our tour starts with a Tayloristic view, where the workers’ efforts to get a useful workplace are almost invisible. Studying the topic from the different research threads in CSCW, we uncover more and more facets of this phenomenon. These facets show the artfulness of situated workplace

design as integrating and managing the coevolution of diverse resources coming from different contexts.

2.1. Tayloristic workplace design

In the field of Information Systems theoretic models about adoption of technology in organizations have been suggested from a positivistic stance (e.g. Fichman 2000; Venkatesh and Davis 2000). Empirically these models are typically validated with the help of standardized surveys. In addition, there is a growing literature on using a practice lens to study the appropriation of technology in organizations (e.g. Orlikowski 2000; Boudreau and Robey 2005). However, there are only few ethnographic studies that examine how workers deal with the design of their workplaces in order to get their work done.

This lack of research might also be an outcome of the implicit assumption of a Tayloristic view that workers should not design their workplaces themselves. Following Taylor's (1911) principles of Scientific Management this issue is in the duty of the management and should rest on the expertise of system analysts. Despite the fact that these principles rarely appear in pure form, Jirotka et al. (1992) point to the fact that they

> *"permeated deeply into management philosophy and appear to form part of the background assumptions of many of those who design computer systems for organizations"* (Jirotka et al. 1992).

With regard to the design of computerized workplaces we found such permeation in the common standards for IT management like ITIL and CobiT (van Bon et al. 2004). These standards describe a set of 'best practices' including the provisioning of IT services and the maintenance and operation of IT infrastructures. The provision and configuration of IT systems is thereby a part of common IT services, which are carried out by service providers. The service provider can be the internal IT department, or outsourced to an external partner. The ITIL standard does not address the single user as the customer of an IT service, but the organization as a whole. Therefore it is not surprising that the topic of tailoring is not addressed within the ITIL standard (neither by end users alone nor cooperatively with the service provider).

2.2. From taylorism to tailorability

A central demand of Participatory Design was the democratization of work; giving the end user a voice in the workplace design (Ehn 1990). In the beginning Participatory Design focused mainly on giving the end user a voice at the early stages of software development projects. Methods like future workshops or mock-up prototyping were used to support the mutual learning between designers and users (Floyd et al. 1989). However, even if design is conducted in a

participatory manner it became clear that monolithic or too inflexible systems cannot cope with the complexity and the dynamics of the world. The democratization of work therefore should include the design of tailorable systems (i.a. Henderson and Kyng 1991; Wulf 1994; Muller et al. 1997). Kahler (1995) emphasized this as "from taylorism to tailorability" to characterize this new workplace design paradigm.

Tailorability comes along with new challenges as anticipating the scope of possible changes (Stevens et al. 2006); considering the variety of tailoring skills of an heterogeneous group of users (MacLean et al. 1990); and bridging the gulf between surface and deep customization (Bentley and Dourish 1995). Tailorable systems should therefore follow a gentle slope of complexity allowing the user to use broader tailoring features step-by-step (MacLean et al. 1990) and keep a reasonable trade-off between ease-of-use and degree of freedom (Costabile et al. 2006).

A way for implementing stepwise increasing freedom and complexity is to provide three levels of tailoring (Henderson and Kyng 1991; Mørch 1997). These levels can be defined as follows (cf. Mørch 1997): *customization* (or parameterization) as modifying attribute values by selecting among set of predefined configuration options; *integration* (or composing) as linking together modular pieces of functionality by script mechanisms (e.g. macros recorder) or by plugin mechanisms (e.g. extension managers); and *extension* (or programming) as adding new functionality by changing existing program code or develop new modules.

To demonstrate the technical feasibility of tailorable systems several systems have been developed in research. Examples are OVAL (Malone et al. 1995), Prospero (Dourish 1996), DODE (Fischer 1994) or FreEvolve (Wulf et al. 2008). These design studies showed that systems could provide tailoring options at different levels of complexity. However, most of them served only as research prototypes that were never used in practice. Accordingly no ethnographic studies exist about how end users use and tailor these systems in the wild.

2.3. Patterns of sharing customizable working environments

A large body of literature has considered tailoring to be an individual effort (cf. Paetau 1991) and research on tailoring support therefore mainly focused on personalization (i.a. Kobsa and Wahlster 1990; Oppermann 1994). This focus shifted in reaction to empirical studies that uncovered the collaborative dimension of tailoring (i.a. Mackay 1990; MacLean et al. 1990; Gantt and Nardi 1992; Wulf 1999; Kahler 2001). These studies raised awareness of the existence of sharing habits and different types of users who are involved in these processes. To categorize the different user types several similar classification schemes have been developed, e.g. the one of Mackay (1990). Following her, we can distinguish between: *lead users* of new technology as users who intensively look into new software and create and share adaptations with others; *translators* as less technical oriented users, who connect the lead users to ordinary users by

relying on the work of lead users and adapting this to the users needs; *ordinary users* as people who do no adapt themselves but use adaptations of other persons.

The existence of different user types and their collaboration seems to be a general phenomenon that is not limited to the organizational domain. In the domestic domain, for example, Grinter et al. (2005) found that the party with the biggest technical competence usually configures home IT for the others. In the general domain of adopting individual products the classical Diffusion of Innovation (DOI) also identified different types of users including innovators, early adaptors, the majority, and the laggards. There are several similarities between both classification schemes. However, there is also an interesting difference: Mackay described lead users as *creators* of innovations *within* the local context, while the Rogers described the *innovators* as adaptors of innovation coming *from the outside*. Therefore they have a slightly different function:

> *"[Most individuals do adopt new products] not on the basis of scientific research by experts, but on the basis of the subjective evaluations of near peers who have already adopted the innovation. These peers [typically innovators and early adopters] serve as models whose behavior is imitated by others in the social system."* (Rogers 2003).

In addition to the empirical research, several authors also exploited opportunities to support collaborative tailoring adequately: Understanding groupware tailoring as a kind of collaborative design process in the small, Oberquelle (1994) argued to support the stages from getting aware and discussing tailoring needs over the evaluation possible solutions to the implementation of one solution and the notification of affected users. Further, Kahler (2001) suggested technical as well as organizational support, including: sharing of configurations and tailored artifacts e.g. via email or built-in mechanisms, curating a repository of tailored artifacts e.g. via a shared file system, enabling the exploration of tailored artifacts in a sandbox, raising awareness of tailoring activities and fostering a tailoring culture including the cooperation among colleagues, the cooperation between users and local experts and the organizational recognition of tailoring efforts.

2.4. From tailoring to appropriation research

In the last 10 years the term *appropriation* appears in CSCW research and since then broadened our understanding about the ways how users give technology a meaning and how they fit technology into the patterns of their everyday life (cf. Silverstone and Haddon 1996; Dourish 2003; Pipek 2005; Balka and Wagner 2006; Stevens 2009). The appearance of the concept was encouraged by phenomena of unanticipated use (Robinson 1993), the situated, cultural

production of meaning (du Gay et al. 1997), and the transformation of work practice in the process of adopting and adapting technologies (Dourish 1996).

Etymologically, the term appropriation is rooted in the Latin word *appropriare*, "to make one's own". Historically the theoretical concept can be traced back to the Marxian/Hegelian evolutionary anthropology (cf. Stevens 2009). The central idea of this anthropology is that man is constituted by labor as the self-realization of man in nature through the appropriation of nature (cf. Márkus 1978; Röhr 1979). Appropriation, in this tradition, refers to the relation between the socio-historically given world and human agency constituting a dialectic unity, where the things we live with only exist within this relation. Appropriation presents an open process of the *situated maintenance and development* of the relation and the boundary between one's own and the foreign. This process has a productive achievement, but is a formative event as well. What a thing is depends therefore on how it is used, and how it appears into human activity. In particular, things itself can change as people change their mode of using it (Ruël 2002).

The salient point in the dialectic view is that *giving things another form* and *giving things another meaning* are not two independent phenomena, but express two facets of solving the challenge to use things constructively, incorporated into one's life for better or worse (Ollman 1971).

Appropriation work (Pipek 2005) as a dedicated activity becomes especially relevant in breakdown situations. During these situations users can try to tailor the features of system as well as explore the existing features in more detail (Stevens 2009). The appropriation work is typically embedded in activities of situated experimentation and explorative learning. It has typically the form of an artful integration or bricolage "using ready-at-hand materials, combinations of already existing pieces of technology—hardware, software and facilities […]—as well as additional, mostly 'off-the shelf' ones" (Büscher et al. 2001).

A facet of Appropriation work is explorative learning. It is closely related (but seldom discussed) to Twidale's (2005) work on the informal, spontaneous workplace help-giving among colleagues who learn to use computer applications according to the local needs. He identified several dimensions to characterize informal learning situations, including: the time, the location, the formality and the topic of the situation. Drawing on the implications for design, Twidale (2000) stressed that in addition to support individual learnability (e.g. by context help) features to support collaborative learnability should also be included into the application. In a similar vein, Pipek (2005) argues for appropriation support, including features to share use experiences and to foster use discourses.

Appropriation research made another important topic visible: the cross-application nature of the people's work. The boundary of a system as intended by designers is often not congruent with the one, perceived and needed by the users when solving the task at hand. Hence, appropriation work is typically accompanied by altering boundaries, re-assembling work materials, and re-configuring organizational, technological as well as spatial relations (Balka and

Wagner 2006). Technology should therefore provide an *outer-tailorability* (Pipek and Kahler 2006), that enables the selection and combination of technologies coming from different sources.

Providing and managing the cross-application tailorability is also a serious topic in the evolution of IT-infrastructures:

> *"Changes—independently of whether they are implemented by tailoring or by evolving the software—can depend on and affect changes in other applications of the IT-infrastructure and the interaction between applications. This requires coordination between tailoring and development, and cooperation between the persons responsible for tailoring and developing the different applications. And this, in turn, requires a different set of competences from users and developers."* (Eriksson and Dittrich 2009).

The artful integration of materials coming from different contexts should therefore be studied from the background, how the coevolution of these materials is organized at the large scale.

2.5. Managing the coevolution of artifacts within software ecosystems

The term software ecosystem refers to a new software production paradigm in Software Engineering (Messerschmitt and Szyperski 2005). In the past, development efforts of software companies were described rather static and individual. Instead, the new paradigm emphasizes the dynamic character of a network of multi agencies. The autonomous actors function as a unit and have to interact with each other either directly or mediated e.g. through market processes.

The software ecosystem paradigm raises new challenges to be solved: the design of software architectures that enable the deep and seamless integration of software components (Bosch 2009; Bosch and Bosch-Sijtsema 2010a); methods to specify the basic architectural structure that (implicitly) defines what is fixed and what is adaptable or extendable (Dittrich et al. 2006); mechanisms to coordinate globally distributed software development (Crowston et al. 2005; Bosch and Bosch-Sijtsema 2010b); dealing with the collaboration and competition at the same time (Henkel 2004; Jansen et al. 2009); and building and managing a software ecosystem around a product including a community of external developers, domain experts and users (Bosch and Bosch-Sijtsema 2010a).

Additionally, there is an increasing awareness that the active role of users has to be considered more seriously. For example Bosch (2009) argues to provide effective mechanism for facilitating mass customization. Yet the topic is still under investigated. In particular, Software Engineering has to consider at a deeper level that developing software will increasingly be mixed and interlaced with the tailoring (Dittrich et al. 2006; Eriksson and Dittrich 2007; Dittrich et al. 2009).

Part of the design problem is the difficulty to anticipate changes for which to provide (Dittrich et al. 2006) and—with regard to open software ecosystems—to

anticipate the participating actors. Another challenge to cope with the multi agencies of software ecosystems is to align common goals with issues of particular interest. The wickedness of "designing for change" cannot be solved in advance, but is in an ongoing accomplishment that requires a continuous communication between software engineers, local experts, and ordinary users (Dittrich et al. 2006). Beyond an organizational tailoring culture, therefore, a cross-boundary culture of participation is needed (Eriksson and Dittrich 2009; Fischer 2009). In addition, Andersen and Mørch (2009) identified five interrelated activities in the coevolution of software: *adaptation* of the product to a specific customer, *generalization* of new release that is available to more than one customer, *improvement request* articulated from the customers' perspectives, *specialization* created in-house to improve the products for their own internal work, and *tailoring* made by end users for their purposes. These activities constitute a system of mutual development and the role of tailoring activities has to be understood from within that system: "Tailoring is better conceived of as evolutionary design, in the sense that the local (customer) solution serves as a design for the general (company) solution, assuming it is accepted" (Andersen and Mørch 2009).

2.6. Local production of large-scale technologies

Another thread of research on the local production of large-scale technologies is given by the studies on 'infrastructuring' (i.a. Bowers 1994; Star and Ruhleder 1994; Karasti et al. 2006; Pipek and Wulf 2009). In a narrow sense infrastructures are large-scale technical systems that are deeply integrated into society. Examples are the telephone system, the railroad system, or electricity. In a broader sense an infrastructure also covers the norms, routines and practices by which the technical system becomes deeply integrated into society. Dropping the idea that infrastructures have an essential substrate, but asking instead when and how to infrastructure, Star and Bowker (Star and Bowker 2006) focused on local practices by which infrastructure becomes a salient, stable resource of action. They call these situated activities of creating order 'infrastructuring'.

Through these practices infrastructures are usually invisible and taken for granted:

> *"Something that was once an object of development and design becomes sunk into infrastructure over time."* (Star and Bowker 2006)

Only when routinized actions become inhibited (e.g. in reaction to a power blackout) practices that were before taken for granted become visible and improvisational recovery work becomes a dedicated activity.

Star and Ruhleder (2001) further explored infrastructures in their quality of decentralized evolving technologies. They uncover a tension between local, customized, intimate and flexible use on the one hand, and the need for global

standards and continuity on the other hand. This tension cannot be resolved once and for all, since

> *"One person's standard is in fact another one's chaos"* (Star and Ruhleder 2001).

Managing the field of tension of local/global and flexibility/standardization, respectively is instead an ongoing accomplished that is manifested in the concrete practices of infrastructuring.

From this stance, Star and Ruhleder (1994) have studied the local context of using the Worm Community System (WCS), a collaborative system for biologists to support sequencing of genetic structure. Similar to the appropriation studies, they observed that getting the system up and running covers a variety of activities that typically become invisible in a standardized description of technology adoption as finding out about the system, installing it, and learning to use it. Bowers (1994) described similar effects for the complexity of work to make a network work. He visualized the unanticipated work that requires users to integrate technical infrastructures into the local context. He notes that there is no unique way to deal with this issue. The significant extra work is not always recognized by others and can even be a reason for abandoning technologies or certain courses of action.

The temporal scales of infrastructure and infrastructuring were further enriched by the work of Karasti et al., who studied the data management within the NSF funded LTER network on long term ecological research (Karasti et al. 2006; Karasti et al. 2010). The central goal of LTER is to promote synthesis and comparative long-term studies across independent research sites. One strategy was to make it mandatory to share the "raw" field data within 2 years of collection. In addition, a long-term oriented information infrastructure was established, where data sharing, data and meta-data standardization, curation and stewardship are necessary, ongoing activities to maintain the long-term usefulness of data. This maintenance is a complex, socio-technical endeavor. Further, Karasti et al. (2006) recognized that the infrastructure in general targets towards a long life span, while the actual infrastructuring activities are often dominated by a short-term perspective. This creates a field of tension between short-term and long-term concerns people have to manage. Therefore, Karasti et al. (2006) argued to supplement the spatial focus of Star and Ruhleder by a temporal scope. According to this, an infrastructure occurs when the tension between local/global and short-term/long-term is resolved, when here-and-now practices are afforded by large-and-long scale technologies, which can then be used in a natural and reliable ready-to-hand fashion.

2.7. Discussion

The users' work to make things work is often invisible and recognized as part of daily work. Further there are efforts in Tayloristic approaches that users should

not have the responsibility to design their workplaces. In contrast, because of the situatedness of work as well as from the normative stance of work democratization there are good reasons to replace the Tayloristic workplace design with tailorable workplace design.

Demonstrating the technical feasibility of radical tailorability, several research prototypes have been built in CSCW. However, because of the experimental character of these prototypes, we know very little about the usage of radically tailorable working environments, in the wild. To cope with this problem, we have to consult the research on customization practices of less sophisticated, but used-in-daily-life applications. They enrich the picture demonstrating that tailoring is not just an individual activity, but has a collaborative quality. The results were enriched by the appropriation studies, which revealed the close entanglement of designing and using workplaces; showing the embeddedness of tailoring in the explorative learning of what an application provides. A similar picture is drawn by the research on infrastructuring. This research thread highlighted that artful integration has to manage a field of tension between the here-and-now of the local context and the once-and-there of decentralized evolving infrastructures. This view was complemented by research that understood production and appropriation of technology not as separate spheres of existence but rather as mutually constitutive of one another. Hence, we have to consider the local practices within their function in the loosely coupled system of mutual development.

Parallel to these threads of research (and partially enforced by them) there is an ongoing trend from monolithic software applications to applications assembled from multiple, coevolving resources of software ecosystems.

From the outlined research, we can make the educated guess that it will become an emerging topic for CSCW to support the *fluent and seamless meshing* of individual, cooperative and organizational practices (Schmidt 2000) of designing workplaces by managing coevolving resources coming from global software ecosystems within the local context.

3. Methodology

Since the 80ties there is a growing market of tools In the software industry supporting the various tasks like compiling, debugging, code control, etc. (Alan 1991; Chikofsky 1992). But using the diverse tools together was clumsy and error-prone. Permeated by tayloristic thinking, in the 90ties the CASE paradigm arose to solve the serious obstacle by the idea of integrated working environments (Bergin 1993) combined with the automation and standardization of work processes (McLure 1989). However, despite great efforts in design and research the paradigm failed to realize the promises (Sommerville 2006) and did not reach acceptance of the practitioners (Elshazly and Gover 1993; Juhani 1996; Lending and Chervany 1998).

From a CSCW stance the failure of the CASE paradigm might not be as surprising as it sounds like the story about the rise and fall of the Office Automation program (Schmidt 2011). However, what makes the story interesting is the change of the paradigm and the nowadays existing universal tool platforms like Eclipse that are open for a growing tool market. In particular, Eclipse has become one of the dominating working environments for software developers in the last years.

In addition, we decided to investigate in Eclipse as kind of leading domain (Von Hippel 1986) that provides favorable conditions to study the emerging practices of designing workplaces by using the new opportunities of open software ecosystems in the wild. Methodologically, we studied the appropriation of the Eclipse ecosystem with the help of a mixed method approach (Kelle 2001):

To answer *question #1* we took a closer look on the development rhythm as well as coevolution mechanisms of the Eclipse ecosystem at the large scale. This work is mainly based on studying existing literature and online documents about Eclipse. Our knowledge was further shaped by personal experience and talks with various Eclipse stakeholders (committers, participating companies, and representatives of the foundation).

To answer *question #2* we conducted an online survey from February 2008 until April 2008. The online survey consisted of a questionnaire, which additionally asked the participants to add certain Eclipse installation data. This allowed us to analyze which plug-ins had been installed by the participants. The study was announced in different online forums, mailing lists, at our project partners and in two research institutes (however to protect the anonymity it was not possible to determine which respond came from which context). We addressed several different target groups of the Eclipse user community (computer science students, software professionals, project leaders etc.). The survey asked for information on the local Eclipse installation, which gave insights into the features and plug-ins the users had installed. In addition we asked how often they adapt their configuration, in what setting Eclipse is used and how the people stay informed. 138 persons participated in the survey and 59 additionally sent us their Eclipse installation data, which we analyzed in detail. Surprisingly, we received 76 sets of Eclipse installation data for our analysis, because some persons were using more than one installation. This also means that these users own more than one Eclipse installation on their computer.

To answer *question #3* we conducted an ethnographically oriented case study about the appropriation of Eclipse in the organizational context of Alpha, a small software company. Previous research on collaboration tailoring (Mackay 1990) and technology adoption in general (Rogers 2003) mainly focused on patterns observed in the social network. In contrast we addressed this topic from a slightly different angle. We took the situatedness of appropriation work more serious and therefore investigated in more detail into the diverse situations that constitute appropriation work.

The study was part of a publicly funded research project on component-based end user development. In the project, we cooperated with different software companies in Germany—one of them being Alpha. We had a special and trustworthy relationship to this company grounded on a close cooperation with a software project, where Alpha took over the source code written by us. Because of this, we visited Alpha and met the developers several times at their workplaces to support their understanding of our source code. We discussed open topics during visits, by telephone or email. One disadvantage of such a kind of ethnographically informed study was our strong engagement during our site visits. This left almost no time for field notes. However, this setting also had certain advantages. We were e.g. not perceived as outsiders. Instead, a collegial atmosphere among people who work together on a task characterized our meetings. This personal relationship was very helpful to gain profound insights into the specific context.

In addition, together with some master students who wrote their thesis on the topic of Eclipse appropriation, we interviewed five persons from Alpha (the CEO, one senior developer and three junior developers). The interviews were semi-structured and took about one hour each. They covered questions about the role, the tasks and the responsibility of the interviewees in the company. In addition, we asked questions about their experience with Eclipse as well as their update and learning strategies. And finally, we searched for ways to improve the diffusion of tools and tool-expertise in the company.

All interviews were recorded, partly transcribed and analyzed together with field notes. While interpreting the context, we made use of our personal knowledge that is grounded on the close relationship with Alpha. We supplemented these by again analyzing selected pieces of the empirical data in detail, by applying the principles of the Objective Hermeneutics (Oevermann et al. 1987).

The structural ecosystem analysis, the online survey and the ethnographical oriented investigation, triangulate the phenomena from alternate points of view. The structural analysis outlines, how the ecosystem works at the large. The survey visualizes from a bird's-eye view some general patterns about users' adaptation behaviors, yet without the concrete context. Hence, in order to understand how people design their workplaces as part of their daily work, the case study follows the advice of Livingston (1987) to move, metaphorically, the camera to eye level.

To answer questions #A, #B and #C, we analyzed the case especially with regard to identifying support opportunities. The aim is to envision possible futures of designing digital workplaces grounded in the empirical material. Methodologically, the link between our empirical studies and design considerations is not a causal, but an inspirational one. Further, we try to uncover the links between the Eclipse case and existing literature, in order to conclude by analogy, how existing approaches could be adapted to the new possibilities given by software ecosystems and modern software architectures.

We want to close this section with a general methodological remark: Like Grounded Theory and Ethnomethodology, the Objective Hermeneutics is a reconstruction-logical methodology (Bohnsack 2003). It is guided by interpretation principles such as the immanent, extensive and verbatim interpretation of records that follow the sequential structure by applying the principle of austerity. The aim is to reconstruct the practical accountable orderliness of the social world as it is expressed in the concrete situation of practical action and practical reasoning (Livingston 1987; Pilz 2007).

The literature on reconstruction-logic approaches shows a common agreement to

> *"remain sensitive to the data by being able to record events and detect happenings without first having them filtered through and squared with pre-existing hypothesizes and biases"* (Glaser 1978).

However, there is a general methodological dispute about the role of previous knowledge and the—in our opinion—too restricting advise that researchers should not read related literature until the end of an inquiry (cf. Kelle 2005). Our position in this regard is that we should not subsume the phenomena under existing categories taken from somewhere else (e.g. form related literature). Yet, a profound knowledge about related literature is often quite helpful to see through existing categories.

To give an example: In an early stage of our research we reconstruct from our an interview transcripts that one fundamental action problem need to solve for all practical purposes is to manage the field of tension between having a stable working environment, while keep up date with the technology developments at the large scale (a detailed documentation of this analysis is published in Schwartz 2007). At that time we did not know the work of Karasti et al. (2010), who in parallel also found that balancing this tension is a serious issue of infrastructuring work. Yet, even if we knew this work beforehand, our analysis would not have become less "grounded". In contrast, knowing Karasti et al.'s work might have helped us to get aware more previously that we uncover a phenomenon that is of general interest for CSCW.

With regard to the collaborative dimension we studied, the situation was slightly different. In this area we had a profound knowledge about the previous research on collaborative tailoring and appropriation. In addition we had a partial knowledge of Twidale's studies on collaborative learning. This sensitized us to take closer look on that topic. Yet, we would insist that the observed orderliness of the concrete practices itself lead us to the categories outlined in Section 6.

4. Eclipse as a global ecosystem

The case of Eclipse is in several dimensions an example for a global software ecosystem. Each of them is worth being studied for its own sake. With regard to our research interest we concentrate on three relevant facets: the growth of a large-scale ecosystems; the architectural strategies at the large to cope with the spatial tension

between flexibility and standardization; and the socio-organizational practices at the large to cope with temporal tension between reliability and innovation.

4.1. Transformation of eclipse into a global ecosystem

Eclipse, with all its historical contingencies can be described as the transformation of internal solutions of the problem on how to integrate a heterogeneous network of product development divisions into a global informational production ecosystem (cf. O'Mahony et al. 2005), where a distributed development process has to be coordinated (Grinter et al. 1999). IBM started the story of Eclipse in the 1990s as an answer to several internal and external challenges. In the mid-1990s, IBM shifted its strategy to a software- and service-oriented enterprise. The IBM Software group had grown rapidly, also by the fact that IBM had acquired a large number of other software development companies. As a result, IBM's software portfolio was only loosely coordinated. This led to several problems of 'inter-usability' (e.g. tools did not have a common 'look and feel',) and inter-operability (e.g. it was difficult to exchange data among the applications). IBM was also confronted with the problem that the applications had been independently developed from the beginning and that components could not be shared in order to save costs. As a result of this organizational context, the idea of Eclipse as a common integration platform for several software tools was born. It was planned as a coordination strategy (cf. Grinter et al. 1999) to manage the loosely coupled production and product network inside the firm. Extensibility was a critical design decision: IBM and its partners wanted to integrate different modules and applications seamlessly.

The next step in the history of Eclipse was related to IBM's middleware strategies, which consisted of three parts: the applications—built by ISVs, the application-development tools (like IBM Visual Age, Sun's NetBeans or MS Visual Studio) and the server software (the cash cow in the strategy of IBM). In order to convince ISVs to adopt Eclipse and to send out a clear signal not to lock out developers on a proprietary platform, Eclipse was made an open-source product. An egalitarian Eclipse Consortium (now the Eclipse Foundation) was founded. All members of the consortium were to have equal decision rights:

> "*[W]e created this dual edged or bi-polar organization that on the one side would play by Open Source rules of engagement to develop the technology and of the other side was the eco-system side, or the commercialization of the technology.*" (O'Mahony et al. 2005).

Today, Eclipse is a multi-facetted brand with millions of users. Eclipse stands for example for a platform technology (e.g. the whole Lotus product line is based on Eclipse) that is available on multiple operating systems (including Mac, Windows, Linux and others), for the second most used IDE today, for an Open-Source project, for a standard-like consortium (organized in the Eclipse Foundation, supported by big players like IBM, SAP, Oracle, etc.), for a software ecosystem (where ISVs built

more than 1000 different extensions and applications on top of the Eclipse platform) and/or for an ecosystem (where an Open-Source community co-exists with commercial players). In addition, commercial products like ondemand.yoxos.com or poweredbypulse.com are specialized in maintaining repositories of 3 rd party plug-ins for Eclipse and supporting organizations as well as end users to pick up plug-ins from these repositories in a safe manner.

4.2. "Everything is a plug-in": the technological fundament of an ecosystem

Eclipse is a living software ecosystem that faces the problem of a consistent evolution of the heterogeneous network of producers and products. The strategy Eclipse implements to provide consistency can mainly be studied from a structural and process perspective.

On the structural level, Eclipse applies an 'everything is a plug-in' philosophy (Beck and Gamma 2003) to address the requirements of flexible and extensible infrastructure. This means that Eclipse is decomposed into hundreds of components (so called plug-ins), which use features of other plug-ins themselves and provide extension points to be used by other plug-ins. Through this component architecture, an Eclipse installation is technically specified by the acyclic dependency graph between the plug-ins of the installation.

In the first two versions, Eclipse was based on a proprietary component model, but since version 3 it is based on the industry standard "OSGi". OSGi defines a sophisticated component model supporting independent loading mechanisms, dependency resolving, versioning control, etc. This architecture is to protect components from corruption by others and to address the integration problem at the same time. In particular it manages situations where two components are used by a third component (but in a different version).

The component networks do not only create dependency graphs in a technical sense, but also in an organizational sense, i.e. between different actors in the Eclipse ecosystem. This means the component architecture is a technical as well as a social artifact. Therefore, the component architecture also affects the power structure and negotiation processes inside the Eclipse ecosystem, as changes of plug-ins included in the core distribution have a greater effect than changing peripheral plug-ins, distributed by 3rd parties:

> *"You need someone who can be a strong advocate to protect the integrity of the platform; you need someone who has the strength to say: 'no we are not going to put that in the platform if it is only for your tool.'"* (O'Mahony et al. 2005).

An interesting aspect from an End User Development research perspective is how Beck and Gamma translate the Eclipse plug-in philosophy into a discourse of empowerment that is based on the idea that designers should

> *"[g]ive the users an empowering computing experience and provide learning environments as a path to greater power"* (Beck and Gamma 2003).

Based on this idea, they argue that the plug-in concept constitutes a pyramid of increasing commitments and rewards, in which the committers of the Eclipse Foundation are at the top. In the middle of the pyramid are publisher and enablers, who contribute third-party plug-ins to the Eclipse Ecosystem without being part of the Eclipse core. End users are also part of the game, as they build the bottom of the pyramid. They can influence the design of Eclipse directly by configuring and extending their Eclipse installations. Since we take a CSCW and HCI perspective on Eclipse, the view of these end-users at the bottom of the pyramid, constitute our field for research.

4.3. The "Eclipse Way": the rhythm of evolution

On the process level, Eclipse has to face the challenge of providing a stable and consistent network of plug-ins and simultaneously innovating it. One of the major problems in this process is that the further development of one piece in the global plug-in network can lead to defects in other parts. The only secure method to prevent this is to stop any changes, but this also hinders the innovation and reaction to dynamics in the environment. Unlike this draconic solution, the Eclipse strategy (sometimes called *The Eclipse Way*) is to create as much transparency as possible, and to establish a generally accepted evolution rhythm, so that independent production processes can be synchronized with each other. The transparency helps Eclipse core projects as well as third parties to stay aware of changes (e.g. through API or plug-in refactoring) and project progress. In addition, the transparency allows users to give feedback at early stages to influence further developments.

The heart of the Eclipse evolution is a specific development rhythm. It is structured as follows: 12 months pass between every major Eclipse release. This time is split into different phases: *warm-up* (1 month), several *milestone builds* (9 months) and *endgame* (1–2 months). The warm-up and milestone phase are innovation-oriented and allow for new features to be implemented. All milestone goals are released in form of a release plan at the Eclipse foundations website, as well as the resulting milestone builds themselves, which was announced with a *"news and noteworthy"* description in order to foster community feedback. The endgame phase is stabilization-oriented and consists of continuous switches between integration, testing phases and bug fixing phases. In the endgame, different release candidates are published (like 3.2RC6). Each release candidate is more stable than its predecessor, ending in a new major release (like 3.2).

In addition, public nightly builds and integration builds are created. Their target groups are users and developers who are eager to figure out the quality of the integration of the components they use or develop and to detect integration

problems. Supporting the integration work on the producer network side is important for global quality management.

4.4. Discussion

Summarizing the background of Eclipse, we can describe it as an evolving socio-technical network, where technical dependencies between individual plug-ins are negotiated between different actors in the environment of related socio-economic dependencies. The Eclipse Foundation—which is a non-homogenous organization, a political institution of different interest groups—presents the center of the network. Dealing with the problem of how to organize the global evolution and integration of an independently produced, but inter-dependently operating network of products, Eclipse applies innovative professional strategies: on the structural level, the plug-in concept helps to establish trust in the beneficial nature of the existing technology among the different stakeholders in the network. On the process level, the strict evolution rhythm with the transparency strategies helps to establish similar trust in the beneficial nature of its future technology among the different stakeholders in the network.

The analysis shows that the spatial and temporal dimensions outlined by Karasti et al. (2010) are also relevant the software production. In particular, agree with Karasti et al. that a structural analysis of large-scale ecosystems is only complete, if it investigates both dimensions. The main difference between studies like Bowers (1994), Star and Ruhleder (1994), or Pipek and Wulf (2009) and our case, however, is that the software architecture of Eclipse is better prepared to integrate resources coming form different actors (Beck and Gamma 2003) and the existing Eclipse configuration manager provides some rudimentary mass-customization features (Bosch 2009). With regard to the several feedback processes between design and use, the mutual development concept (Andersen and Mørch 2009) seems an interesting candidate. However, the concept should be extended with regard to the multi-organizational character of Eclipse and the institutionalized government structure of the Eclipse Foundation.

5. A Survey on eclipse appropriation

In this section we present the findings of the online survey. The major goal was to find evidence of the work to make things work by managing coevolving resources coming from different contexts (Bowers 1994; Balka and Wagner 2006). By its nature this artful integration is almost invisible and difficult to quantity (Star and Ruhleder 1994; Dörner et al. 2008). Existing research mainly relies on qualitative evidence for the existence of users' integration work. In order to quantify it, we use the tailoring at the level of integration (Mørch 1997) as a proxy to measure users' integration work. Of course, this proxy captures a small fraction of the whole phenomenon, but it nevertheless is helpful to get an

impression. On the other hand, every feature in Eclipse is a plugin, and therefore tailoring at the level of integration is mainly expressed by adapting Eclipse at the level of plugins. Hence, in our survey we focus on plug-ins.

5.1. Adapting eclipse as a regular activity

As a first step, we were interested in how many plug-ins are used in practice. This should help us to answer several questions (1) how complex is the appropriation task users are confronted with in their efforts to manage the Eclipse ecosystem, (2) is the modification of Eclipse installations a common practice and (3) what do Eclipse users usually modify.

We were surprised to find 2,428 different plug-ins within the collected sample (the number rises to 4,944 when we take the different versions into account. This means that on average each plug-in was installed in two different versions). The average number was 326 plug-ins per installation.

Furthermore, we analyzed the so-called *features* of the captured Eclipse installation data, as these are the basic elements of update management and installation management in Eclipse.[1] The concept of features reduces the complexity of the plug-in network for the users. Instead of managing about 326 plug-ins, the user only has to manage around 40 features (cf. Table 1). The standard deviation of features σ_f=36.8 is an indicator for the diversity individualizing Eclipse. Furthermore, we calculated the normalized average distance between two Eclipse installations. The value of $\bar{u}_{feature}$=0.42 confirms the findings of other empirical data, which stated that practically no Eclipse installation resembles another one.[2]

Regarding the integration of a heterogeneous network of producers, we tried to find out, if Eclipse is used as an off-shelf product or if 3rd party plug-ins from independent ISVs are integrated into Eclipse installations. We therefore focused on features that are not delivered by the Eclipse foundation. One of these features is the support for the *Subversion* source-code version control system for Eclipse, which was by this time provided by two different independent open source projects. At the time of the survey, none of these tools were integrated into Eclipse by default; instead it is up to the user to integrate this extension into the Eclipse installation if Subversion support is needed. In our sample 40% of the Eclipse installations included Subversion plug-ins, which is a strong indicator that the users make use of the global market of Eclipse extensions.

Table 1. Amount of plug-ins found in Eclipse installations (with n=76 Eclipse installations).

Overall number of features (no versions counted)	418
Overall number of features found (version sensitive)	865
Min. number of features in an Eclipse installation	3
Max. number of features in an Eclipse installation	196
Average number of features per Eclipse installation	42
Standard deviation of features per Eclipse installation	36.8

In order to learn how the evolution of Eclipse is reflected in the installation data, we took a closer look at the version number of the core feature *org.eclipse.platform* (which is part of every Eclipse installation). In our data, we found 11 different versions. 60 installations are of the 3.3.X release (published June 2007), 12 cases of the 3.2.X release (published June 2006) and 3 cases of the 3.1.X release (published June 2005). We did not find an installation based on one of the Eclipse 3.4 milestone builds, released a few weeks before the survey (which we expected after our workplace study). Within the range of 3.3.X releases, 36 cases were not older than 2 months. On average, a version in use is approximately half a year old.

In addition the online survey asks several questions on the practices to integrate the global plug-in network into the local context. In a first step, we were interested whether the adaptation of Eclipse is a common and regularly practice. Therefore we asked: *"How often do you adapt your Eclipse (installation and update of plug-ins, or configuration settings)?"*. Almost all participants (92.66%) declared they would adapt their installation to their needs (7.34% never, 14.71% right after the installation, 77.21% sometimes, 0.74% daily). This result corresponds with the analysis of the installation data. In addition, it shows that adapting the working environment is not only a singular, but in most cases a regular activity.

5.2. Local network of eclipse users

We were also interested in strategies that inform people about activities of the Eclipse ecosystem, the role of collaboration and installation sharing practices. In particular, we were interested in seeing whether a local network of Eclipse users exists. Therefore, we asked: *"How many of colleagues of you also use Eclipse?"* The majority (71.32%) explained that in local environments also other persons use Eclipse (only 4.41% say no other person uses Eclipse, 24.27% give no answer to that question). This confirmed our workplace observation of existing local social networks of Eclipse.

5.3. Getting tools and tool information

We also asked: *"How do you inform yourself about new plug-ins?"* The most frequent answer was the Internet with 78.48%, colleagues were mentioned by 54.43%, 21.52% use magazines and 6.33% use special online plug-in marketplaces (multiple answers were possible). This demonstrates that the Internet as a global resource is the most used source for information, but also it demonstrates that local social networks play an important role.

The question *"Do you have ever received plug-ins from colleagues?"* also addresses the aspect of collaboration, but directly focuses on the diffusion of plug-ins. The answers also indicate that local social networks play an important role in the appropriation of the global network of plug-ins (65.44% of the participants stated 'yes', 17.65% stated never and 16.91% gave no answer).

Furthermore we were interested in the channels used for the diffusion of plug-ins, therefore we asked: *"Which ways did you use to receive these plug-ins?"*. Figure 1 gives an overview of the answers (it was possible to choose multiple answers). The answers demonstrate that there is not just one way used for plug-in diffusion. However, 69.41% of the Eclipse user state that they receive plug-ins via personal communication and 32.94% say that in some cases they have used a file copy strategy to get the plug-in on the desktop. Both answers are a indicators that local networks also play an important role in the diffusion of plug-ins, although this was not anticipated by Eclipse designers and although it is not well supported by Eclipse.

The analysis of the online survey shows that the dynamics of the global Eclipse evolution and the heterogeneity of the Eclipse plug-in universe are reflected at the micro-level of Eclipse installation. It also demonstrates that local social networks play an important role in the appropriation of global Eclipse network of loosely coupled components.

5.4. Discussion

Eclipse is one of the most advanced technologies today. It's architecture in combination with a living software ecosystem enables users to design their workplaces by assembling tools from different vendors. The survey gives quantitative evidence that this not just as a theoretically given option. The given answers as well as the returned installation details shows that almost every user adapted his Eclipse configuration to his needs. This finding emphasizes the significance of integration work mentioned in literature (Star and Ruhleder 1996; Balka and Wagner 2006). Moreover, the survey demonstrated to what extend adapting Eclipse became part of ordinary work activities and ordinary work situations.

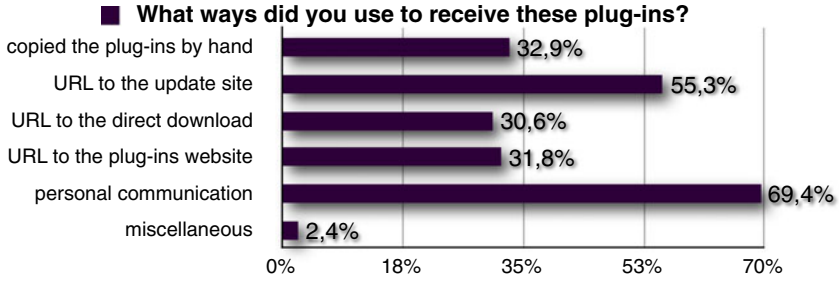

Figure 1. Channels used to receive plug-ins and plug-in information. The other way round *"Did you ever share plug-ins with colleagues?"* and *"Which way did you use to share these plug-ins?"* provide nearly identical pictures.

The survey also shows that colleagues are an important resource to get information about plugins as well as plugins itself. The findings are in line with the Diffusion of Innovation theory (Rogers 2003) as well as the research on collaborative tailoring (Mackay 1990). However, neither Rogers nor Mackay discuss that people share digital 3rd party goods. With regard to the observed patterns, we should therefore have in mind Eclipse' open source character that legalizes this gift culture.

6. Appropriating eclipse in an organizational context

In this section, we present the findings of the in depth case study at the small software development company Alpha. Our primary goal was to understand and describe the structure of appropriation work that is carried out at the shop floor. We expected to learn about, how people get aware of new tools, how they learn to integrate them into their work places and how they learn to use these tools and related methods.

Taken the situatedness of appropriation work seriously, we especially focus on diverse situations that constitute appropriation work. The Table 2 presents a list of diverse types of situations we found is. These situations were relevant for the collaborative appropriation, yet we did not claim that this list was exhaustive by any means. Before we discuss the diverse situation in detail, we start with a general description of Alpha to help the reader to grasp the context of our results.

Table 2. List of situations that contextualized appropriation work.

Team meetings	Institutionalized auditorium to discuss tools in the whole group. The meetings are typically co-located and conducted regularly.
Shared infrastructure breakdown	Work to fix problems of the commonly used IT-infrastructure. These situations are not planned, but occur in reaction to an accidental event. Typically, the effects are distributed and cover an assembly of tools and IT systems.
Looking over the shoulder	Observing new tools or tool usage by working together or by chance encounters. Typically, these are ad-hoc peer-to-peer situations, where the people are co-located. Usually they are embedded in actual work situations.
Giving a jump start	Joining a new team or project. In order to save the new person from unnecessary preparation work on his own, whole working environments are copied or information about the used tools is given. Typically these peer-to-peer situations are embedded in work situations where people are co-located.
Getting contextualized help	A new task or problem during work drives a person to ask colleagues that are for some reasons considered more experienced in this topic. These are peer-to-peer, ad-hoc situations that occur embedded in work situations. Mostly co-located.

6.1. Organizational context

Alpha is a small software company, which is quite typical for German software industry. The company was founded 20 years ago as a spin off to commercialize *AlphaProduct,* a web application that was developed during a research project. The application was implemented by making use of the Python programming language and was extended by several Java applications. Today, the web application presents the main asset of the company. Alpha's market strategy is based on selling AlphaProduct licenses and creating customer-specific adaptations. Furthermore, they are regularly involved in research projects, in which the application is constantly extended by innovative features.

The company permanently employs about ten persons. In addition, there are a small number of university students, working part time for the company. The customer relations are mainly in the duty of the CEO, who is supported by assistants that accomplish general administrative chores and office work. The software development carried out at Alpha, deals with the maintenance and the continuous enhancement of AlphaProduct. This work is mainly conducted by eight software developers.

The development work at Alpha is organized as projects that can be categorized as follows: The first type involves client projects that are carried out to realize customizations or new features for a specific customer. Typically, client projects have short durations. Usually, there are 2 or 3 developers involved, depending on the complexity of the tasks. The second type covers projects that realize innovative features, which are typically conducted as a part of funded research projects. These projects are typically larger as for the amount of work, are long running and are carried out with other partners.

In client projects, the CEO typically serves as the interface between the costumer and the software developer. The CEO obtains the wishes and requirements of the client and discusses them internally with the developers in order to create an offer. At project start, the staffing depends on the actual workload of the workers as well as on their general expertise, prior knowledge and experience to work the job. The staffing of the project decides, to a large extend, how the project is decomposed into individual work packages and how work tasks are assigned. The work is coordinated mainly by communication. In addition, also other mechanisms like implicit coordination by using a shared repository are applied:

> *"....well, during my studies I have come to know eXtreme-programming to manage things. Here, we set things up so that we can divide everything up into sub-projects…we divide projects into sub-projects, that means into special areas so that there will be a specialist for each area…and every specialist works in his area of the project. […] The specialists do of course exchange implementation ideas …and we have also introduced a common repository."*
> (John, Junior Software Developer)

If possible, developers are assigned to certain tasks based on their previous experience. This promotes the formation of knowledge niches, meaning that the developers become specialists in one part of the application (e.g. adapting the data layer, implementing the user interface or writing Java applets). However, the overall rule in the company is that "whatever is necessary has to be done" (Paul, Junior Software Developer). This can lead to the situation that developers have to work on tasks, even if they are not specialists for this. Through this, they also gain knowledge in other parts of the application.

6.1.1. *Laissez-faire management*

Taylor (1911) argued that standardized tool equipment is needed for certain tasks and that only the management will be able to determine this set of tools for the best results. This basic assumption is still reflected by Information Systems standards such as CobiT and ITIL (van Bon et al. 2004) as they make the tasks of tool selection and provisioning the business of the management. Yet, at Alpha we observed that every employee is allowed to freely choose his/her tools to work with. As a result, the workplace installations of the developers are quite heterogeneous. In particular, the management gives its blessing to this autonomous working style:

> *"People that have been brought up with Unix and vi and Emacs and stuff, they have a hard time dealing with it [Eclipse]. They've worked with it once in a while but don't really see its benefits for themselves. And feel more at home in their environment. And I don't tell people how to do their work as long as they get it done."* (Peter, CEO)

We asked the CEO whether the heterogeneity of the workplace installations does not increase the complexity of their working together. His position on this subject was that the cooperation among the team members is reached not through tools, but through discussions, working on the same source code and using the organizational bug tracking system.

6.1.2. *Dissemination of eclipse*

This freedom or autonomy also affects the adoption of Eclipse. In the company a camp of younger developers as well as the CEO adopted Eclipse, while the older generation constituted of senior employees (about 40 years and older), does not use Eclipse. The older generation was socialized by old-fashioned Unix systems, using textual consoles rather than graphical user interfaces and a set of mostly command line tools. They "grew up" with those tools and workflows and feel more at home with their existing situation. From their point of view, Eclipse has its merits, but is generally not perceived as a beneficial tool:

> *"We once discussed it [the use of Eclipse]. Actually, the main point of criticism is the slow operational time vs. that of a simple text editor…still features are quite interesting. Especially the integrated version management. Seeing how you don't need an additional external tool … that has something to it."* (Peter, CEO)

The young generation of employees is constituted of two students (about 20 to 25 years old) studying computer science at the local university. They work part time at Alpha. Both are socialized with Windows and feel more at home in a graphical user interface environment than in command line environments.

The first student works about 10 h per week at Alpha. Within the company, he is employed in software development work. He calls himself the Java and Eclipse expert. He has known Eclipse for about 8 to 9 years now (since Version 2.1) and sees himself as an experienced user. The second student works about 2 days per week at Alpha. His tasks are software developments with Python and Java. He describes himself as a mature Eclipse user. Both know Eclipse from programming courses at the university.

In addition to the younger developers as well as the CEO use Eclipse. The CEO holds a degree in computer science and knows both the old and the new tools. He has used Eclipse for 8 years and regularly joins software development tasks. He did not portray himself as an expert or power user when it comes to Eclipse, but he appreciates it as a toolbox capable of integrating quite different tools, and offering a common look and behavior over the whole range of tools.

For newcomers who join the community of practice (Wenger 2007) at Alpha, the laissez-faire management creates opportunities to bring new software engineering methodologies and tools into the company. This especially holds for Eclipse as their preferred working environment. In this case, they serve as innovators, while old-timers could profit from their expertise. Such a case of switched roles, regarding learning activities, was described by the CEO:

> *"Well, you know Paul recently downloaded and installed some files for editing JavaScript files. It was from Aptana. There is this bundled Aptana studio version and one that could be installed into Eclipse as individual plug-ins…. and since I needed it too, I've asked him about the plug-in version before installing it. As it also looks so nice, but you know, he told me that it wasn't that good. […] So I downloaded the Aptana Studio…which is a good alternative to it. Apart from that I don't have that many special plug-ins running."* (Peter, CEO)

6.2. Situations of collaborative appropriation

Throughout our study, we observed or found traces of appropriation work, contextualized by situations. Analyzing these situations in addition to the

practices that had been carried out, helped us to understand the structure or practices of appropriation work at Alpha. But furthermore this focus on appropriation situations helped us to understand the constituting situational context that renders certain observed practices useful or not.

In the following subsections, we present detailed information on these situations of collaborative appropriation work.

6.2.1. *Team meetings*

One opportunity to share appropriation experiences with each other is the team meeting that takes place once a month. In this meeting, all developers at Alpha get together and discuss the current state of running projects and topics that occurred during day-to-day work. Tools and their usage or management are not regular topics. However, we found different incidents that the issue of tools becomes important enough to be discussed during the meeting.

For example, the company is dependent on certain technologies like the Python interpreter used by customers to run AlphaProduct. This forces Alpha to take care of the development plan and release-rhythm of the Python project. In particular, if the currently used interpreter version is going to be deprecated in the near future and if one employee of Alpha gets to know about this, he will make it a topic in a team meeting.

Another example would be to explore technological options in order to realize emerging requirements like the integration of AJAX features into AlphaProduct. Sometimes one of the developers is asked to carry out an inquiry about tools and technologies on the market. If the results of the investigation turn out to be interesting for the others, they are presented and discussed during the meeting.

6.2.2. *Shared infrastructure breakdowns*

The separately used, but tightly interwoven tools and technologies, constitute a *shared infrastructure* in an organization (Pipek 2005). This infrastructure evolves in a decentralized manner through the situated activities of the people. In the case of Alpha, the mentioned Python interpreter is part of the shared infrastructure. Each employee should use the same interpreter when developing, testing and debugging AlphaProduct to prevent incompatibilities at the customer's site. Other critical systems are the ones that are used by all developers like the bug tracking system or the version control system.

In principle, modifications made on the shared infrastructure should be coordinated in order to prevent breakdowns. However, because of hidden dependencies, it is not always easy to judge what modifications could lead to a breakdown situation. Therefore, activities to prevent breakdowns are not coordinated beforehand, but in reaction to dealing with an occurred breakdown situation (Pipek and Wulf 2009).

The chance of breakdowns is increased by the laissez-faire management, which does not regulate the workplace design, but leaves it to the individual's choice. In the case of Alpha, we observed dealing with a breakdown that occurred during the regular maintenance of the Subversion server, which is the version control system used in the company. We recognized this when we visited Alpha and someone told us that we can't connect to their subversion system. We were told that the responsible person updated the repository service software. Unfortunately, this was incompatible with the clients in use and caused a breakdown for several developers. Later, during an interview, we recognized that since everyone who had noticed the problem asked the maintenance person, he was quick to search a fix for the problem. He found that a certain version of the repository client would enable his colleagues to work again and thus spread this information. After the rest of the developers updated their clients, they could use the repository again:

> *"then all the others installed it [the subversion client] and then it finally worked again. You see, based on these problems, we started communicating."* (John, Junior developer)

The shared infrastructure breakdown constraints the laissez-fair rule that the workplace design lies within the individual's authority. Moreover, the breakdown was one of the sparse situations where the software developers adapted their workplaces collectively.

6.2.3. *Looking over the shoulder*

One important type of informal situation of the collaborative appropriation that we observed is what we call "over-the-shoulder appropriation". We adopted this term from Twidale's (2005) concept of *over the shoulder learning*. Originally, the concept describes forms of informal learning in organizations that happen in over-the-shoulder situations. The case of Alpha, however, shows that the core concept is suitable to describe certain forms of collaborative appropriation too.

When working together, it happened occasionally that one developer got aware of a new tool or trick just by looking over his colleagues shoulder:

> *"If we now sit down on something and work on it together…we can borrow from each other. Features like key combinations or if we see that someone uses a plug-in we don't know about yet."* (Peter, CEO)

In order to become aware of new things, experience will have to be shared. Because of the shared context, situated and tacit knowledge could be used in the over-the-shoulder setting to explain special features and demonstrate tricks of how to use the tool. As demonstrated by Twidale (2005), one typically took over the role of the teacher and one took over the role of the learner. However, as we have seen in the example of the CEO who received advice from a newcomer, the roles were not necessarily ascribed according to their status as a senior or junior developer. In

addition, the roles could also switch during the situation. An essential point is that the appropriation effect emerged spontaneously from within the situation and was not intended beforehand. We therefore speak about opportunity based appropriation, which is fundamentally different from other forms of informal situations like *asking for help* or *jumpstarting*, as we will describe below. This is not just an analytic difference made by us as researchers, but also creates a difference in practice:

> *"You know it wasn't like I actually went down there and had him show me his (Eclipse] installations and stuff...Like only when we are working together in front of his PC I say..hey man, this is a new icon. What's that?"* (Peter, CEO)

But over-the-shoulder situations are not only important for the dissemination of tool expertise. They are also important for the dissemination of the tools (the plug-ins as an artifact) themselves. Either in an over-the-shoulder situation or later, as a reaction to the situation, tools were exchanged and integrated into the personal working environment. However, the fact that the collaborative appropriation is not intended does not mean that a person is aware of the opportunity when sitting together with someone else:

> *"I am still curious to look over someone's shoulder. You know, I peek interestedly at what is now written there on the title bar…even if those are totally unknown applications to me."* (Peter, CEO)

6.2.4. *Giving a jump start*

The work preparation is a typical situation to appropriate new tools. In opposite to the ordinary flow of work where tools are ready-to-hand, in such situations the tools and their installation become present-at-hand. We found a collaborative form of this kind of appropriation situation in cases where someone new joins the team. In such situations the members of the already existing team introduce the new colleague to the current tasks, problems, infrastructure that is used, etc. Additionally, the individual working styles are coordinated among the team members. This covers issues as for applied code conventions, strategies for code integration etc. but also for the installation of the workplace.

An innovative practice we found to cope with this challenge is that a team member gives a *jump start* to the newcomer. In other words, the newcomer receives all of the needed tools and tool information to be able to directly start working in the team. This saves time and prevents errors through configuring the environment in a proper way by himself. Through jump starting, the newcomer finds an environment that is adjusted to the situated project context and that also helps to prevent breakdowns of the shared infrastructure as described above. In the case of Alpha, one of the people, who in the past gave jump starts, is the CEO:

> *"If someone is new or joins a project …we have some tools we use like the plug-in for Python and for Subversion and most often I send off emails or tell people*

where they can find these and they can arrange their own stuff…or otherwise everyone is free to make adjustments and to extend their things." (Peter, CEO)

6.2.5. *Getting contextualized help*

The user's small, individual breakdowns in the flow of work also play a critical role in the appropriation of software systems. The remarkable aspect of breakdown situations is the user's switch in focus. The concerns around the original work task are pushed into the background, while the reflection on tools and methods to get the work done is being initiated. This reflection phase now often results in asking colleagues for specific help. We found different initial situations that resulted in specific requests for help. Sometimes users had installed an incompatible update, in other cases they faced a new task that required a special tool that was not yet within the personal toolbox. Such situations have in common that they refer to a *"need driven"* (Peter, CEO) appropriation. This is in contrast to the *opportunity given* appropriation, which we defined as becoming aware of new things, e.g. in the mentioned over-the-shoulder situation or by reading in a magazine or website about new developments in the Eclipse ecosystem.

One of the major problems of need driven appropriation is to obtain a market overview. Since there exist hundreds of vendors, it is quite difficult to find the exact solution for a problem. Therefore, to ask colleagues for help to enhance the view on the market is a standard practice at Alpha. Only if this is not successful, one must start the quest himself:

"Well the thing is…I want to have a plug-in for something…and if someone tells me it is this plug-in, the whatever it is called, and it solves your problem. … Then I try to get the name of the plug-in, I will google it and then I have no trouble finding it—fortunately. But if there is no one else in the company that knows about it, then of course I myself have to search for it." (John, Junior developer)

Asking colleagues rather than Internet forums comes with another advantage. Quite likely, Alpha employees share the same context. This again eases the expression of the problem and results in better advice. Especially, since the advice giver is maybe an experienced user of the tool in question:

"Before I bother to somehow download it [a plug-in] and then realize that I need something else and then spend half a day to install it…to realize that it doesn't meet my requirements…I rather meet up with someone of whom I know he might already have used it or that he still uses it… I ask for his experience with it, and if he tells me it is super or alright, then I would follow his example…if he says it's crap, I would drop it." (Peter, CEO)

A key challenge the people have to cope with when appropriating new tools is to innovate while keeping the work going. Yet, getting familiar with a

new tool involves a lot of time. Furthermore, it may be uncertain if there will be improvements of the work practice at all. Instead, the existing workplace may be corrupted. In our interview, the CEO gives a quite good illustration of the related problems and the role of asking colleagues to deal with them:

> *"It's about efficient usage of time or even about saving time. It also is a trial. I think that for some it's also always an obstacle…new tools also require learning curves…that is to say I must make someone else provide me with it and install it,* etc.*…..and then I have to be able to manage it and learn how to use it…and have to see how it works and to see what I can do with it…and finally realize that either all my requirements have been met so that it makes work easier…or I realize that it is not what I have been looking for…then I've spent hours and hours for nothing….and that I could already be done with it. That's why I think that many don't take the trouble to get it or that they cannot do it because they lack time….to simply try something out is often not possible. So you are happy if someone else has already experienced this."* (Peter, CEO)

In general, we observed that the advice of colleagues is perceived to be more important than advice found in magazines or the Internet. However, this varies depending on the personal taste as well as the specific situation. In addition, if colleagues cannot help, searching the web or magazines is the usual fallback solution:

> *"If I know that someone makes use of it on a regular basis, and if I have problems with it but he could know it…then I will ask him about it…or don't even hesitate to ask the developers' community about the plug-in. Often they have mailing lists."* (John, Junior developer)

6.3. Discussion

The case study helped us to understand how the members of Alpha make use of the possibilities to adapt their Eclipse configurations and which situations contextualized their actions.

One of its merits was to get a closer insight in the new competencies that users need because of the rising interdependency of tailoring and development (Eriksson and Dittrich 2009). Our research revealed that one of the new competencies is to keep the personal skills and tools up to date with the development of the ecosystem. This is a general infrastructuring competency (Star and Bowker 2006; Karasti et al. 2010) to manage the spatiotemporal tension between the local and the global context. The need and the specific form of the competencies are shaped by the dialectic between the specific structuring of the software ecosystem at the large and the work practices at the shop floor. At the

large, users have to deal with a decentrally organized Eclipse ecosystem (see Section 4). In particular there is no general quality standard of the tools, as they come from quite different sources as manufactures and hobbyists. In addition the market of tools is quite dynamic and furthermore fragmented as no centralized distribution instance exists (an example for such an instance would be Apples App Store). Hence, part of the specific competence is to keep track of a fragmented market as well as dealing with the uncertainty that integrating a new tool could corrupt the entire working environment.

Another key result of this study is to show that designing the workplace by making use of software ecosystems is not a competence of the individual user, but a collective competence of the workgroup or whole company. This collective competence is maintained in various situations like regular team meetings, break downs, asking for help or introducing juniors to a new field of work.

Taking care about the own working environment is further closely linked to the *habitus* of software developers. Strübing (1992), described this habitus rather as a craftsmanship than assembly-line work. He concluded that software developers tend to stay on the *bleeding edge of technology*, as it is perceived as a part of their professional ethics. Our case study complements his finding, but in this case showed the co-existence of two different camps of software developers. The first camp mainly consisted of older developers. They retained the old style of command line oriented programming, as they rely on their well-proven tools. In opposite, the second camp, mainly consisting of younger developers, followed an *always be up to date* or "bleeding edge of technology" attitude as described by Strübing. However, both camps commonly felt responsible for their work tasks and their tools.

In general, the case study also confirms the findings of the survey (Section 5) and the previous research on collaborative tailoring (Mackay 1990; MacLean et al. 1990) or technology adoption in general (Rogers 2003). In particular we saw that local networks and interpersonal communication are part and parcel of the appropriation of new tools and methods. In addition, we see different types of user with their specific roles in the process of adopting and disseminating new technologies. For example, we observe that users collaborate by acting as experts (by sharing knowledge and giving advice) as well as by sharing appropriation artifacts (e.g. components or preference settings) when modifications are necessary.

However, the roles are not static, but could vary between situations as well as within a situation (Twidale 2005). In particular, a general assumption in the Community of Practice (CoP) theory is that newcomers learn form the old-timers (Wenger 1999). However, our study nuanced picture concerning the adoption of new tools and the learning of the usages. An example of an opposite case as outlined in the CoP is the CEO, who adapted a whole new toolset from the junior developers. Murphy-Hill and Murphy (2011) mentioned a similar case, showing that seniors can learn very much from junior developers. Their explanation is that

juniors have more time at their hands to play around with bleeding edge technology.

Getting aware of new tools or tool usages often happens in informal situations based on peer interaction. A typical example for this kind peer interaction the is "over the shoulder" situation described by Twidale (2005). Our case study demonstrates that in this kind of situation learning, configuring, and adopting are closely entangled. All these activities constitute the appropriation as a whole (in the outlined Marxian/Hegelian sense). We therefore argue to extend Twidale's concept by the notion "Over the shoulder appropriation", where we have to study the different parts within their function of appropriation.

A common assumption is that people start to learn a new tool in reaction to a concrete need. Yet, in our case study we have also seen that people play with new tools "just for fun". Taking the "infrastructuring" perspective (Karasti et al., 2006; 2010), the function of this kind of appropriation is defined by satisfying a latent need to resolve the temporal tension between the actual work practices and the innovation development at the large scale and in the long run. In particular, innovations are not adopted because there is a need, but also because there is an opportunity. In our work we therefore distinguish "opportunity driven" and "need driven" appropriation.

Good opportunities to appropriate a new tool or tool usage are over-the-shoulder situations. However, in advance one cannot ensure that a tool, satisfying an existing need, can be found by accident. Need driven situations are therefore differently structured. We found two examples of such kinds of situations: The situations of contextualized help and the situations of shared infrastructure breakdowns.

Getting contextualized help is typically carried out by interaction with near peers. Murphy-Hill and Murphy (2011) discussed "over the shoulder learning" and "getting contextualized help" as one category, named peer-interaction. Through this lens they got a fine-grained analysis of the diverse peer interaction situations, where people talked about tools and tool usages. This is a valuable addition to our findings. However, they did not distinguish between concrete (need) and latent (opportunity) needs.

In particular, satisfying a concrete need has typically a much higher priority than satisfying a latent need, as one is under time pressure. The most prominent example of satisfying a concrete need in our case study was the failure the common used version control system. This situation of a shared infrastructure breakdown (Pipek 2005) brings the users' attention to reflect the current infrastructure and furthermore enables him to think of actions to fix the breakdown. We have seen that fixing the breakdown was not an individual need, but a collective and a collaborative effort was necessary to fix the problem for the involved persons.

By highlighting the peer interaction and the ad hoc situation, we should not downsize the institutional situations for the collaborative appropriation. Team

meetings, for example, serve as a platform within companies for knowledge sharing and work coordination. Therefore we wanted to categorize these situations as the natural arena to discuss appropriation efforts. And we were surprised to find, that previous work had not described team meetings as platforms for appropriation work. However, there is a note in the literature on agile methods, that one standard topic in the daily meetings should be

"What problems are preventing me from making progress" (Pipek 2005).

Projecting our findings that tools play an important role in making or not making progress, we expect the daily meeting also to serve as a platform for appropriation work. However, this has to be explored by empirical work.

Another category we haven't found in the existing research on appropriation and related topics was the situation of "giving a jump start". This activity is some related to sharing tailored artifacts as well as knowledge sharing fostering organizational learning. However in the scope of software workplaces and their appropriation it has never been discussed. Yet we found that it was the mix of sharing information and readily tailored artifacts that constitute a good jump start.

7. Some futures of supporting the appropriation of software ecosystems

The previous section has outlined that the openness of software ecosystems is double-edged: while it introduces a new freedom, it also comes with new burdens. Bosch (2009) has stressed mass-customization as an appropriate strategy to keep the advantages of software ecosystems, but lower the burden for the users to make use of it. Bosch does not further elaborate this thought, but typically mass-customization is understood as means for individualization focusing on the individual user only (Franke and Piller 2003). The CSCW research on collaborative tailoring as well as the case study, however, demonstrated the shortcomings of this design philosophy: instead of designing for the individual user, a design that integrated the needs at the shop floor has to be created. From this stance we will discuss in the following sections support opportunities at three levels: The personal level, the level of the local team and the organization and at the global level of the ecosystem.

7.1. The personal level

The personal level addresses the appropriation work of the individual user, as he adapts and adopts tools from the Eclipse ecosystem. An important aspect of the individual appropriation work is to *safely* manage the different plug-ins of an Eclipse installation. Eclipse already includes several features to support this issue. First of all, its software architecture itself is an enabling technology. It is designed to make plug-ins coming from different vendors work together. In addition, the

integrated configuration manager (see Section 4.4) provides a build-in mechanism to download, install and remove components.

Despite these support mechanisms we observed several workarounds that users developed because the support provided by Eclipse was not sufficient. In the process of adoption (Rogers 2003), we can analytically separate the issues of keeping the tool competence up-to-date, keeping the tools up-to-date and managing multiple installations. In practice, however, they are typically interwoven. Therefore, a design concept should cover these activities in an integrated manner.

7.1.1. *Keeping the tool competence up-to-date*

Adapting the system is just one part of appropriation (Dourish 2003). In particular, before installing new tools one has to be aware of their existence. In addition, one has to know their purpose and one has to learn, how to use tools effectively. To support these activities, we should distinguish two cases:

- Keeping the competence up-to-date with regard to a tool that is already in use and
- Keeping the competence up-to-date with the general development in the plug-in market and the general software engineering field.

With regard to the first case, a solution should make users aware of available updates for tools they already use, e.g. by a service that regularly performs checks in the background. Such a mechanism is already provided by Eclipse. And as long as the manufacturers of a plug-in follow certain specifications, it works quite well. This mechanism on the other hand lacks to present information that describes what an update is good for. In principle, updating existing tools means that a user does not have to learn to use the tool from scratch. Instead, migrating to the new version is mainly determined by the question: *"what has changed since the last version?"* Hence, in the context of updating a tool this kind of information should be given.

With regard to the second case our study reveals that we should distinguish between *need driven* and an *opportunity driven* appropriation. Need driven appropriation refers to activities as searching solutions for a specific problem, e.g. searching a plug-in to integrate the bug-tracker into the working environment. As seen, users typically ask colleagues for help, use general purpose search engines like Google or make use of special web sites (see Section 4). From a user's point of view, the search mechanisms should also be integrated into the working environment. This provides the opportunity to use information about the actual installation, in order to only show plug-ins that are compatible and/or recommended.

Opportunity driven appropriation refers to situations, where people keep the tool competence up-to-date through regularly reading magazines, studying web sites or becoming aware of interesting tools by looking over a colleagues shoulder. However, supporting opportunity driven appropriation

can only be supported in a heuristic manner. We will discuss this issue in more detail in Section 7.2 within supporting the tool awareness among colleagues.

7.1.2. *Keeping the tools up-to-date*

After becoming aware of either a new tool or an update to an existing tool and the decision that this tool/update might be worth a try, it must be integrated into the Eclipse installation. Technically, this means the user has to modify the Eclipse installation, including all issues of transferring and installing plug-ins as well as mechanisms to recover the installation in the case of a breakdown.

Several issues are supported by Eclipse. However, the functionality is split into several user interfaces like a dialog to update installed plug-ins, a dialog to enhance the workplace by installing new plug-ins. Another dialog allows for inspecting, and disposing plug-ins as well as recovering older states of the installation.

Furthermore, Eclipse does support the exchange of tools among team members (see Section 6.2.4). Because of this, the user must change to the file system level to install plug-ins that he copied from a colleague.

Following the easy-to-adapt principle of End User Development (Lieberman et al. 2006), the functionality could be improved by integrating the diverse features into one tailoring environment that could directly activated form the use context (Wulf and Golombek 2001).

7.1.3. *Managing multiple installations*

Appropriation processes do not just add to an Eclipse installation. Instead, managing multiple installations and removing plug-ins is just as important. A first reason to use more than one installation is the skepticism that the made modifications are reliable. Because of this, some developers do not just remove outdated installations, but keep different old versions as kind of fallback system. For example, we observed a developer, who always creates a backup copy, before installing new plug-ins. After an adequate period of testing, he copies the fallback system to a folder containing older installations, which are outdated. Then he uses the testing environment as his new productive system.

Furthermore, because Eclipse installations are adapted to the specific demands of a project, one might want to be able to access them later. Even if a project had been finished and one turned towards a new project, there was still a possibility that a bug had to be fixed or the customer requested new features. In such case one can benefit, if the old installation is still available and can be reactivated.

A third reason is that people sometimes work on more than one project. Because of this they want to be able to switch quickly between different toolset installations.

One drawback of the outlined Eclipse installation manager is the missing support for having multiple installations. Instead, the design rests on the underlying assumption that a user has only one Eclipse installation, ignoring above usage scenarios like using explorative and fallback installation or having special installations for each project.

Because of these shortcomings, users have created their own workarounds. Examples are installing Eclipse for each project in separated folders, manually backing up Eclipse installations at file system level or disposing plug-ins at file system level that caused a breakdown. The existing literature addresses these issues partially by introducing the idea of exploration environments. These should allow users to try adaptations in a sandbox and revert if anything went wrong (Kahler 2001; Wulf and Golombek 2001). Yet the case study showed that the need to explore is not the only reason to create backup copies of software work places. Another reason is the need to work on different projects in parallel that may need differently configured tools. Therefore we need a broader concept than exploration to support the practices we observed.

Commercial solutions like Yoxos or Pulse are more sophisticated regarding the management of multiple Eclipse installations. In particular, they offer the opportunity to specify and name different installations - so called *profiles*. This simplifies the management of software portfolios and the selection of the right installation for the task at hand. However, at the moment these solutions do not respect certain organizational decisions (e.g. having a set of tools as base) and specific demands of a project or personal favorite addition. As a result, it is awkward to maintain the diverse profiles when personal preferences or organizational constraints did change. In addition, features are still missing to define exploration environments (Wulf 2000), which allows to experiment with new tools in a safe manner.

7.2. Local level of the organization

Appropriation work inherently has a collaborative facet (Pipek 2005). Furthermore, at Alpha we found a kind of informal collaboration (Mackay 1990). This also holds true in the case of sharing Eclipse plug-ins. As we have seen, sharing plug-ins is often rooted in personal contacts who work in the same project. Furthermore, we observed that cooperation could even cross-organizational boundaries, as people interact with the community or external workers, as students, who join a project team.

The finding that people in local context give advice to install, update or try something new also holds for the case of appropriating the Eclipse ecosystem in the organizational context. But differently to the work of Mackay (1990), innovators are usually not the creators of the modification. Instead, they are usually the first adaptors of a new tool that was built by someone else. This feature refers to the work of Rogers (2003), who described the process of social systems, adopting externally

developed innovations. However, he reflected on the adoption of individual artifacts only and did not take the users creativity and the complexity into account that arises when dealing with multiple artifacts, coming from different sources, at once. In addition, he did not further investigate into the structure of the situations where forms of collaborative appropriation could be observed.

With regard to this, the merits of our case study are not just to demonstrate that social networks are important for the appropriation of software ecosystems. In Section 6.2 we further outlined the structure of the related situations that are constituted by collaborative forms of appropriation.

An interesting approach to support opportunity driven appropriation is to address online over-the-shoulder situations. Awareness mechanisms for tailoring activities as described by Kahler (2001) look like promising approaches. As an example, notifications could be used if something important happens within the organizational network. The design of this awareness support additionally should be tightly integrated in the user interface (Twidale 2000; Pipek 2005; Stevens and Wiedenhöfer 2006).

To support need driven appropriation, search based user interfaces and recommender techniques seem promising. Concepts like the Expert Finder (Reichling et al. 2009), originally developed to support knowledge management, are also a promising approach for collaborative appropriation. They can be used to identify tool expertise in the organization and the personal social network and draw connections between experienced and advice-seeking users.

In addition in the case of a collective need affecting the whole team (which is sometimes subtle to answer), the collaborative tailoring mechanisms outlined by Oberquelle (1994) seem to be promising to make the collaborative negotiation and realization processes more effective.

While the different forms of appropriation result in different requirements on the level of user interaction, the required data and data gathering mechanisms are very similar for both forms. A promising approach to gather the needed data is to trace changes of the Eclipse installations within the team. Even more detailed information can be provided by tracing usage histories, as this leads to more precise results (see also Pipek 2005). Additionally, we can enable users to rate and comment plug-ins to provide advice that can be accessed later.

A further general requirement to support collaborative appropriation is to enable the sharing of plug-ins among team members and colleagues from within the working environment. At the moment, users are forced to copy plug-ins on the file system level, which is awkward and error prone.

A first research prototype that realizes some of the mentioned requirements is Peerclipse (Draxler et al. 2009). The intention of Peerclipse is to respect the habitus of the developers being responsible for their tools, but at the same time to support the collaborative appropriation among the team members. Peerclipse is integrated into the working environment and

establishes a local peer-to-peer network support awareness of tool expertise and sharing plug-ins with each other.

7.3. Global level of the ecosystem

Existing research on tailorability focuses mainly on the local context. However, our case demonstrated that individuals and groups who tailor Eclipse are at the same time linked to the global community. This underlines Eriksson and Dittrich's (2009) remark that Kahler's (2001) work has to be adapted to mutual development of situated tailoring and software development in the large. In particular the concept of an organizational tailoring culture should be enlarged and embedded in a culture of participation at a large scale. We therefore want to both discuss the global level of the Eclipse ecosystem as a whole and discuss the way the individual user is connected to it.

The Eclipse ecosystem today consists of millions of users, thousands of developers, dozens of organizations and several thousand software artifacts (see Section 3 and 4). We observed that users who go beyond the scope of their team or organization (e.g. because no one else in the organization tested this new code repository client before), interact with people from the global community as to figure out what the market looks like, how stable is a plug-in, if its features are sufficient and how it can be utilized.

On the global level, there is a good chance that someone very experienced for the current problem exists. Unfortunately, finding these people and fostering the collaboration is only poorly supported today. The global level presents therefore a new challenge and a chance we should consider when designing appropriation support. Through this new dimension the personal and local level do not become obsolete. Instead, they support each other and supporting design concepts need to connect and integrate them.

On the global level efforts like the Eclipse marketplace (a plug-in market) and its integration into the working environment as well as commercial products like Yoxos on Demand or Pulse are highly interesting. They all try to offer to the user a single point of access to all independent vendors and their products. In addition, they try to bring the tool market closer to the use context in order to overcome the separation between distributing digital goods and configuring the working environment. This indicates a transformation of the traditional mediation between an open network of producers and users. However, they do not fully grasp the opportunities to foster cooperation among the ecosystem members.

For example in a further step these systems should visualize available expertise as well as implicit or latent cooperation needs. In particular, they should integrate a social infrastructure closer into the Eclipse application system. Realizing such ideas, we should consider that users typically try to identify expertise within the nearby social network first. If this fails, they try to do the same for the whole

ecosystem. One example of system design that supports such cascading strategies of collaboration is outlined by Stevens (2009).

However, we just began to explore the interplay of local to global collaboration and further research is needed to support a smooth transition between the different levels.

8. Conclusion

In the last years, software ecosystems that are constituted by networks of coexisting and coevolving software are grown. This new way of software development is primarily studied from a software engineering point of view, neglecting the consequences for the design of software workplaces. To address this gap we investigated in Eclipse as one most successful examples of the new software ecosystem paradigm. Methodologically, we used a mixed method approach to measure to what extend people make use of new opportunities as well as to understand how appropriation work is structured at the shop floor.

The results of the online survey confirmed the literatures qualitative evidence that people tailor their applications to fit them into the local context. Moreover, our survey also showed evidence that over 90% of the users adapt their Eclipse configuration by integrating plug-ins that come from different places. This indicates that tailoring activities have become part of their everyday work practices.

The survey results furthermore showed that peers and colleagues are important resources to get new artifacts as well as information about what's is going on in the software ecosystem. Similar patterns were mentioned before in the context of collaborative tailoring (Mackay 1990) as well as general technology adoption (Rogers 2003). Because of these similarities, future research on end user issues of software ecosystems should attempt to synthesize both research threads, enhanced by the dimension of integrating coevolving materials (which is neither addressed by Mackay nor by Rogers).

How is appropriation work structured and what situations contextualize this work? To answer this question, we conducted an ethnographically oriented study in a small German software company. The study uncovered the crucial problem that is resolved by appropriation work. Namely to maintain a reliable working environment at the shop floor, while keeping technologically informed with the fragmented developments in the software ecosystem. These results show the deep link between appropriating ecosystems and the general 'infrastructuring' activities (Star and Bowker 2006) studied so far. Both have to resolve the structurally homologue tension between the here-and-now practices and the evolutionary technologies at the large-scale (cf. Karasti et al. 2010). The structural analysis of the Eclipse ecosystem revealed that this tension is also a product of Eclipse's short release cycles. However, Karasti et al. noted the effects of an increased speed of technological change on infrastructuring as

> *"[i]t is a constant battle to keep up with things, to remain current in technology"* (Information Manager quoted by Karasti et al. 2006).

In contrast to Karasti et al., we took a closer look on the collaborative dimensions of this issue. In particular, we systematically analyzed the diverse situations of collaborative appropriation. We identified different types of situations that contextualize appropriation. Some were already mentioned in literature, like the over-the-shoulder-appropriation that generalizes Twidale's (2005) work by stressing on the close entanglement of adaptation and explorative learning. Another example is the infrastructure-breakdown situation, which is closely related by the work of Pipek (2005) work on dealing with breakdowns in general. However, we also found practices like *giving a jump start* that were not mentioned in literature beforehand and are solely grounded in the empirical data.

This work underscores Bosch's (2009) remark that end-user oriented strategies for software ecosystem are needed. However, its seems that the software ecosystem research so far either neglect this topic or primary focused on the individual user only. A key contribution of this work is therefore to correct this view, showing that appropriating software ecosystems is not primary an individual competence, but a collective competence.

Form this stance we envision at different levels the opportunities to support the appropriation work: We outlined, how existing tailoring approaches (Oberquelle 1994; Kahler 2001; Pipek 2005) could be applied to support the appropriation work at the personal as well as organizational level. However, we must go beyond these approaches and integrate an organizational tailoring culture with a participation culture of the ecosystem at the large (Eriksson and Dittrich 2009). We especially made aware of the new role of intermediary parties that can lower the burdens to collaborate between the manifold actors of the ecosystem. The most popular example of such an intermediary party is Apple with its AppStore. Users do not even consider it tailoring anymore if new *Apps* are bought and installed by a single click. In the case of Eclipse, mass-customization toolkits that provide a repository of 3rd party extensions, like Yoxos on Demand or Pulse, become more popular.

Eclipse nowadays is a quite polished version of a software ecosystem. But even here we can observe that the users' role of establishing (collaboration) relations between different actors has to be further explored in order to be supported. In particular we argue that the guiding principle of mass-customization should be replaced by the concept of appropriation infrastructures (Stevens et al. 2010) that provide an integrated customization and collaboration platform.

With all precautions to generalize a single case, we assume that tailoring applications by making use of software ecosystems will become an important issue in general. However, we have to be careful when transferring our findings to other cases. We have to consider three issues:

- At the large, we have to consider that the Eclipse platform is very successful, as it addresses a market with more than a million users and more than thousand contributors. In addition, it follows a rapid innovation cycle. Other successful and dynamic software ecosystems, like Mozilla, as well as Karasti

et al.'s findings indicate that an increased technological change is a general issue. However, we cannot take this for granted. We therefore argue that attempts to transfer these findings should include a structural analysis of the ecosystem in question.

- At the local context, we have to consider that a laissez-faire management practice was applied during our case study. This gives the responsibility for the workplace design to the hands of the people on the shop floor. As discussed is Section 2.1 this is not the usual view on workplace design (at least considering the literature on IT management). Therefore the findings might not be easy to transfer to other contexts. The survey on the other hand (see Section 4) helped us understanding that a wide range of Eclipse users actively adapt their workplaces, even if we do not know their organizational backgrounds. Moreover, from a normative stance of work democratization, we would argue that this case demonstrates that flexibility is possible and useful.
- Additionally, we have to admit that software developers are usually not considered end-users, but trained to solve technical problems. Several studies confirm their habitus as "following technological trends". We should therefore not expect that in other domains users have the same interest in their tools and keeping up-to-date. If we try to transfer these results to other domains, we have especially have to invent a set of methods, techniques and tools that allow less technical skilled users to modify or extend their working environments (Lieberman et al. 2006). In Section 7 we address this challenge by outlining, how the appropriation of software ecosystems could be made easier in the future.

Considering this, further research has to expand in three different directions. First, we should carry out studies in other companies in order to obtain a richer picture of the diverse appropriation practices in organizations. Additionally, we should consider studying different software ecosystems, like Mozilla or Linux, and comparing them to our results. Finally, we have to study the different user types, their different strategies and needs to maintain a reliable working environment at the shop floor, while keeping informed with increasing speed of technological change.

Note

1. A feature in Eclipse defines a set of plug-ins and sub features which must be installed when the feature is installed.
2. We calculated the distance of two installations with the set of features C_i and C_j as follows: $u_{feature}(C_i, C_j) = (|C_i \backslash C_j| + |C_j \backslash C_i|)/(|C_i| + |C_j|)$. Based on this, we calculated the average distance: $\bar{u}_{feature}(C_1, \ldots, C_n) = 1/n * (n-1) * \Sigma_{0 \leq i < j \leq n} u_{feature}(C_i, C_j)$. A value of $\bar{u}$ near 0 means that the different Eclipse installations are almost identical; a value near 1 means that the installations are most different.

References

Alan, S. F. (1991). Case: Using software development tools. John Wiley and Sons, Inc.

Andersen, R. & Mørch, A. (2009). Mutual development: a case study in customer-initiated software product development. *End-User Development*, 31–49.

Balka, E. & Wagner, I. (2006). Making things work: Dimensions of configurability as appropriation work. Proc. of CSCW 2006, ACM: 229–238.

Beck, K. & Gamma, E. (2003). Contributing to eclipse: Principles, patterns and plugins. Addison-Wesley.

Bentley, R. & Dourish, P. (1995). Medium versus mechanism: Supporting collaboration through customisation. Proceeding of Fourth European Conference on Computer-Supported Cooperative Work (ECSCW'95): 133–148.

Bergin, T. J., Ed. (1993). Computer-aided software engineering: issues and trends for the 1990s and beyond, IGI Publishing.

Bohnsack, R. (2003). Rekonstruktive Sozialforschung: Einführung in qualitative Methoden, Utb.

Bosch, J. (2009). *From software product lines to software ecosystems*. Proceedings of the 13th International Software Product Line Conference (SPLC '09): 111–119.

Bosch, J. (2010). *Architecture challenges for software ecosystems*. Proceedings of the Fourth European Conference on Software Architecture: ECSA '10 ACM: 93–95.

Bosch, J., & Bosch-Sijtsema, P. (2010a). From integration to composition: on the impact of software product lines, global development and ecosystems. *Journal of Systems and Software, 83*(1), 67–76.

Bosch, J., & Bosch-Sijtsema, P. M. (2010b). Softwares product lines, global development and ecosystems: collaboration in software engineering. *Collaborative Software Engineering, 1*, 77.

Boudreau, M. C., & Robey, D. (2005). Enacting integrated information technology: a human agency perspective. *Organization Science, 16*(1), 3–18.

Bowers, J. (1994). *The work to make a network work: studying CSCW in action*. ACM: 298.

Büscher, M., Gill, S., et al. (2001). Landscapes of practice: bricolage as a method for situated design. *Computer Supported Cooperative Work (CSCW), 10*(1), 1–28.

Chikofsky, E. J. (1992). *Computer-aided software engineering* (2nd ed.). U.S.: IEEE Computer Society Press.

Costabile, M. F., Fogli, D. et al. (2006). End-user development: The software shaping workshop approach. *End User Development*, 183–205.

Crowston, K., Wei, K., et al. (2005). *Coordination of free/libre open source software development*. Citeseer.

Dittrich, Y., Lindeberg, O., et al. (2006). End-user development as adaptive maintenance. *End User Development*, Springer**:** 295–313.

Dittrich, Y., Vaucouleur, S., et al. (2009). ERP customization as software engineering. *IEEE Software, 26*(6), 41–47.

Dörner, C., Heß, J., et al. (2008). *Fostering user-developer collaboration with infrastructure probes*. Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering, ACM: 48–44.

Dourish, P. (1996). *Open Implementation and Flexibility in CSCW Toolkits*. Ph.D. dissertation, London, UK.

Dourish, P. (2003). The appropriation of interactive technologies: some lessons from placeless documents. *Computer Supported Cooperative Work (CSCW), 12*(4), 465–490.

Draxler, S., Sander, H., et al. (2009). Peerclipse: Tool awareness in local communities. Demonstration on the ECSCW 2009.

du Gay, P., Hall, S., et al. (1997). *Doing cultural studies: The story of the Sony Walkman*. London: Sage.

Ehn, P. (1990). Work-oriented design of computer artifacts. Lawrence Erlbaum Associates Inc.

Elshazly, H. & Gover, V. (1993). A study on the evaluation of CASE technology. *Journal of Information Technology Management*, *4*(1).

Eriksson, J., & Dittrich, Y. (2007). Combining tailoring and evolutionary software development for rapidly changing business systems. *Journal of Organizational and End-User Computing, 19*(2), 47–64.

Eriksson, J. & Dittrich, Y. (2009). Achieving sustainable tailorable software systems by collaboration between end-users and developers. Evolutionary Concepts in End User Productivity and Performance, IGI Global**:** 19–34.

Fichman, R. G. (2000). The diffusion and assimilation of information technology innovations. Framing the Domains of IT Management: Projecting the Future Through the Past,≈, Pinnaflex Educational Resources**:** 105–128.

Fischer, G. (1994). Domain-oriented design environments. *Automated Software Engineering, 1*(2), 177–203.

Fischer, G. (2009). End-user development and meta-design: Foundations for cultures of participation. *End-User Development*. Pipek, V., Rosson, M., de Ruyter, B., Wulf, V. Springer Berlin/Heidelberg. 5435**:**3–14.

Floyd, C., Mehl, W. M., et al. (1989). Out of Scandinavia: alternative approaches to software design and system development. *Human Computer Interaction, 4*(4), 253–350.

Franke, N. & Piller, F. (2003). Configuration Toolkits for Mass Customization. *International Journal of Technology Management*.

Gamma, E. & Beck, K. (2003). Contributing to eclipse: Principles, patterns, and plugins. Addison Wesley.

Gantt, M. & Nardi, B. (1992). Gardeners and gurus: patterns of cooperation among CAD users. Proc. of CHI'92, ACM: 107–117.

Glaser, B. (1978). *Theoretical sensitivity: Advances in the methodology of grounded theory.* Mill Valley: Sociology Press.

Grinter, R., Edwards, K., et al. (2005). The work to make a home network work. Proc. of the ECSCW'05: 469–488.

Grinter, R. E., Herbsleb, J. D., et al. (1999). The geography of coordination: dealing with distance in R&D work. Proc. of GROUP '99, Phoenix, Arizona, United States, ACM: 306–315.

Henderson, A. & Kyng, M. (1991). There's no place like home: Continuing design in use. *Design at work: cooperative design of computer systems*: 219–240.

Henkel, J. (2004). Open source software from commercial firmsñTools, complements, and collective invention. *Zeitschrift f¸r Betriebswirtschaft, 4*, 1–23.

Jansen, S., Finkelstein, A., et al. (2009). *A sense of community: A research agenda for software ecosystems*. Proc of. 31st International Conference on Software Engineering, New and Emerging Research Track, IEEE: 187–190.

Jirotka, M., Gilbert, N., et al. (1992). On the social organisation of organisations. *Computer Supported Cooperative Work (CSCW), 1*(1), 95–118.

Juhani, I. (1996). Why are CASE tools not used? *Communications of the ACM, 39*(10), 94–103.

Kahler, H. (1995). From taylorism to tailorability supporting organizations with tailorable software and object orientation. *Advances in Human Factors/Ergonomics, 20*, 995–1000.

Kahler, H. (2001). Supporting collaborative tailoring PhD Thesis, Roskilde University.

Karasti, H., Baker, K. S., et al. (2006). Enriching the notion of data curation in e-science: data managing and information infrastructuring in the long term ecological research (LTER) network. *Computer Supported Cooperative Work (CSCW), 15*(4), 321–358.

Karasti, H., Baker, K. S., et al. (2010). Infrastructure time: long-term matters in collaborative development. *Computer Supported Cooperative Work, 19*(3–4), 377–415.

Kelle, U. (2001). Sociological explanations between micro and macro and the integration of qualitative and quantitative methods. *Forum: Qualitative Social Researchs, 2*, 5–24.

Kelle, U. (2005). ""Emergence" vs."Forcing" of empirical data? A Crucial Problem of "Grounded Theory" Reconsidered." *Forum Qualitative Sozial Research*, *6*(2).

Kobsa, A., & Wahlster, W. (Eds.). (1990). *User models in dialog systems*. New York: Springer.

Lending, D. & Chervany, N. L. (1998). CASE tools: Understanding the reasons for non-use. Computer: 13.
Lieberman, H., Paterno, F., et al. Eds. (2006). End-user development, Springer.
Livingston, E. (1987). Making sense of ethnomethodology. Routledge & Kegan Paul.
Mackay, W. (1990). Patterns of sharing customizable software. Proc. Of Conference on Computer-Supported Cooperative Work: 209–221.
MacLean, A., Carter, K., et al. (1990). User-tailorable systems: Pressing the issues with buttons. Proc. of CHI 90, ACM Press: 175–182.
Malone, T. W., Lai, K. Y., et al. (1995). Experiments with oval: a radically tailorable tool for cooperative work. *ACM Transactions on Information Systems (TOIS), 13*(2), 177–205.
Márkus, G. (1978). *Marxism and anthropology: The concept of human essence in the philosophy of Marx*. Van Gorcum.
McIlroy, M. (1968). Software Engineering. Report on a conference sponsored by the NATO Science Committee.
McLure. (1989). *CASE in software automation*. Englewood Cliffs: Prentice-Hall.
Messerschmitt, D. G. & Szyperski, C. (2005). Software ecosystem: understanding an indispensable technology and industry. MIT Press Books.
Mørch, A. (1997). Three levels of end-user tailoring: Customization, integration, and extension. *Computers and design in context***:** 51–76.
Muller, M. J., Haslwanter, J. H., et al. (1997). Participatory practices in the software lifecycle. *Handbook of human-computer interaction, 2*, 255–313.
Murphy-Hill, E. & Murphy, G. C. (2011). Peer interaction effectively, yet infrequently, enables programmers to discover new tools. Proceedings of the ACM 2011 conference on Computer supported cooperative work (CSCW '11), ACM: 405–414.
O'Mahony, S., Diaz, F. C., et al. (2005). IBM and Eclipse.
Oberquelle, H. (1994). *Situationsbedingte und benutzerorientierte Anpaflbarkeit von Groupware. Menschengerechte Groupware-Softwareergonomische Gestaltung und partizipative Umsetzung* (pp. 31–50). Stuttgart: Teubner Verlag.
Oevermann, U., Allert, T., et al. (1987). *Structures of meaning and objective hermeneutics. Modern German sociology: An anthology (European Perspectives)* (pp. 352–434). New York: Columbia University Press.
Ollman, B. (1971). *Alienation: Marx's conception of man in a capitalist society*. Cambridge Studies.
Oppermann, R. (1994). Adaptively supported adaptability. *International Journal of Human Computer Studies, 40*(3), 455–472.
Orlikowski, W. J. (2000). Using technology and constituting structures: a practice lens for studying technology in organizations. *Organization Science, 11*(4), 404–428.
Paetau, M. (1991). Kooperative Konfiguration [Cooperative Configuration]. Proc. of DCSCW'91: 137–151.
Pilz, D. (2007). Krisengeschöpfe: zur Theorie und Methodologie der objektiven Hermeneutik, Duv.
Pipek, V. (2005). From tailoring to appropriation support: Negotiating groupware usage. PhD Thesis, University of Oulu.
Pipek, V. & Kahler, H. (2006). Supporting collaborative tailoring. *End User Development*, 315–345.
Pipek, V., & Wulf, V. (2009). Infrastructuring: towards an integrated perspective on the design and use of Information technology. *JAIS, 10*(5), 447–473.
Reichling, T., Veith, M., et al. (2009). Expert recommender: Designing for a network organization. *Learning in Communities*, 139–171.
Robinson, M. (1993). *Design for unanticipated use*. Kluwer Academic Publishers: 187–202.
Rogers, E. M. (2003). *Diffusion of innovations*. Free Press.
Röhr, W. (1979). *Aneignung und Persönlichkeit (Appropriation and Personality)*. Berlin: Akademie Verlag.

Ruël, H. J. M. (2002). The non-technical side of office technology: managing the clarity of the spirit and the appropriation of office technology. Managing the human side of information technology: challenges and solutions, Idea Group Publishing**:** 78**–**105.

Schmidt, K. (2000). The critical role of workplace studies in. Workplace studies: Recovering work practice and informing system design**:** 141–149.

Schmidt, K. (2011). Cooperative work and coordinative practices. *Cooperative Work and Coordinative Practices*, 3–27.

Schwartz, T. (2007). *Praxisgerechte Unterstützung kooperativer Aneignung am Beispiel der Eclipse IDE*. Diplomarbeit, Universität Siegen.

Silverstone, R., & Haddon, L. (1996). *Design and the domestication of information and communication technologies. Technical change and everyday life. Communication by design: The politics of information and communication technologies* (pp. 44–74). Oxford: Oxford University Press.

Sommerville, I. (2006). *Software engineering*. Addison Wesley.

Star, S. L. & Bowker, G. C. (2006). How to infrastructure. Handbook of new media: Social shaping and social consequences of ICTs: 230–245.

Star, S. L. & Ruhleder, K. (1994). Steps towards an ecology of infrastructure: complex problems in design and access for large-scale collaborative systems. Proceedings of the 1994 ACM conference on Computer supported cooperative work. Chapel Hill, North Carolina, United States, ACM**:** 253–264.

Star, S. L., & Ruhleder, K. (1996). Steps toward an ecology of infrastructure: Design and access for large information spaces. *Information System Research, 7*, 111–134.

Star, S. L. & Ruhleder, K. (2001). Steps toward an Ecology of Infrastructure: Design and Access for Large Information Spaces1. Information technology and organizational transformation: history, rhetoric, and practice, SAGE**:** 305**–**346.

Stevens, G. (2009). *Understanding and Designing Appropriation Infrastructures*. Disseration, University of Siegen.

Stevens, G., Pipek, V., et al. (2010). Appropriation infrastructure: mediating appropriation and production work. *Journal of Organizational and End User Computing (JOEUC), 22*(2), 58–81.

Stevens, G., Quaisser, G, et al. (2006). Breaking it up: An industrial case study of component-based tailorable software design. *End User Development*, 269–294.

Stevens, G. & Wiedenhöfer, T. (2006). CHIC-A pluggable solution for community help in context. ACM: 212–221.

Strübing, J. (1992). Arbeitsstil und Habitus—zur Bedeutung kultureller Phänomene in der Programmierarbeit (working style and habitus—to the meaning of cultural phenomena of programming work). Universität Kassel.

Taylor, F. (1911). *The principles of scientific management*. Harper & Brothers.

Twidale, M. (2005). Over the shoulder learning: supporting brief informal learning. *Computer Supported Cooperative Work (CSCW), 14*(6), 505–547.

Twidale, M. B. (2000). Interfaces for supporting over-the-shoulder learning: 33–37.

van Bon, J., Pieper, M., et al. (2004). *IT service management: An introduction based on ITIL*. Van Haren Publishing.

Vasilis, B., Slinger, J., et al. (2009). Formalizing software ecosystem modeling. Proceedings of the 1st international workshop on Open component ecosystems. Amsterdam, The Netherlands, ACM.

Venkatesh, V. & Davis, F. D. (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. Management science: 186–204.

Von Hippel, E. (1986). Lead users: a source of novel product concepts. *Management Science, 32*(7), 791–805.

Wenger, E. (1999). Communities of practice: Learning, meaning, and identity. Cambridge Univ Pr.

Wenger, E. (2007). Communities of practice: Learning, meanings, and identity. Cambridge university press.

Wulf, V. (1994). Anpaßbarbarkeit im Prozeß evolutionärer Systementwicklung, GMD-Spiegel.
Wulf, V. (1999). *"Let's see your search-tool!"—collaborative use of tailored artifacts in groupware*. ACM: 59.
Wulf, V. (2000). Exploration environments: supporting users to learn groupware functions. *Interacting with Computers, 13*(2), 265–299.
Wulf, V., & Golombek, B. (2001). Direct activation: a concept to encourage tailoring activities. *Behaviour & Information Technology, 20*(4), 249–263.
Wulf, V., Pipek, V., et al. (2008). Component-based tailorability: enabling highly flexible software applications. *International Journal of Human Computer Studies, 66*(1), 1–22.