The Appropriation of a Software Ecosystem

A Practice Take on the Usage, Maintenance and Modification of the Eclipse IDE

Sebastian Draxler

Dissertation zur Erlangung des akademischen Grades

Doktor rerum politicarum

(Dr. rer. pol.)

an der Fakultät III:

Wirtschaftswissenschaften,

Wirtschaftsinformatik und Wirtschaftsrecht

der Universität Siegen

Jahr der Fertigstellung: 2014

Erster Gutachter: Prof. Dr. Gunnar Stevens

Zweiter Gutachter: Prof. Dr. Volker Wulf

Tag der mündlichen Prüfung: 23. Oktober 2014

Abstract

This thesis was written in order to gain a deeper understanding of the appropriation of software in groups and organizations. In doing so, it focuses on software created under the modern software engineering trend *software ecosystems*. Software ecosystems have a major influence on software development, as they rely on massive usage of distributed software development, open source models and modularization. It is unclear if existing models to explain appropriation still hold good. Furthermore, it has to be explored whether current appropriation support is still appropriate and beneficial or if we need new ideas to help users cope with these developments.

In order to achieve these objectives, this work is mainly based on an empirical field study, which investigates workgroups at seven German organizations that use the Eclipse IDE, an extremely modularized and adaptable software system, developed by a globally active ecosystem of large corporations, small businesses and even hobbyists.

Using the qualitative analysis approach of the grounded theory method and appropriation as a lens for this research endeavor, observations and interviews as well as artifacts were analyzed, uncovering practices that are part of Eclipse usage and appropriation. They are identified and discussed from the backdrop of software ecosystems – viewed from a users perspective. Examples are the sheer amount of appropriation activities at the shop floor, the dilemma of software maintenance, that comes with continuously developed but sometimes unstable technology, practices as learning or tailoring, influences on practices stemming from the software ecosystem, the organization and the group.

Grounded in these results, suggestions for the design of appropriation support are given and prototypically implemented, which reflect the embeddedness of individuals and groups in the software ecosystem. They provide a fresh perspective, based on peer-to-peer technology and awareness mechanisms.

Acknowledgements

There are a lot of people who have to be mentioned for their importance to me and to this work.

I cannot thank Gunnar Stevens enough, who acted as my mentor for many years and as my supervisor for this work. Working with him has lead to my interest in this topic and in the end to this book. Many ideas and insights of my work are based on the discussions we had during my work on the thesis and our collaborative work for the CoEUD project. Furthermore for his co-authorship and constructive feedback.

I thank Volker Wulf, for guiding my interest towards the intersection of Computer Supported Cooperative Work and Software Engineering and for awaking my interest in research. His advice and support through the whole process was especially important.

Special thanks to Alexander Boden, Kai Schubert and Claudia Müller for their co-authorship, valuable feedback and input during the creation of many publications and the finishing phase of this thesis. Furthermore for coffee.

I like to thank Martin Stein, Hendrik Sander, Adrian Jung and Tobias Schwartz, for being my students in the past and for helpful collaborations.

Rachel Schneider and Oliver Stickel have to be thanked for proof reading of the final version. This contributed a lot to the readability of this document.

I would also like to thank my other colleagues from the University of Siegen and Fraunhofer FIT whose feedback and inspiration has helped.

Also, I thank all the participants of my studies, who remain anonymous in this thesis, for the invitations to their workplaces, for allowing me to conduct interviews and observations, for contributing Eclipse-usage related data. You contributed greatly to the insights embedded in this thesis.

I also feel deep gratitude towards my family and friends (especially Thorsten), who have always supported me morally through the years that it took to write this book; without their support, patience and encouragement, this would not have been possible. Last but not least, I especially want to thank my partner Ania, who supported and encouraged me, despite being in a similar situation. Dziękuję.

Thank you all!

Related Publications

Parts of this dissertation have already been published as conference or journal papers. The chapters of part II and part III resemble the accepted versions of these publications:

- **Chapter 4:** Draxler, S. & Stevens, G., 2011, Supporting the Collaborative Appropriation of an Open Software Ecosystem, Computer Supported Cooperative Work (CSCW), 20, pp. 403-48. With kind permission from Springer Science+Business Media. http://link.springer.com/article/10.1007%2Fs10606-011-9148-9
- **Chapter 5:** Draxler, S., Jung, A., Boden, A., et al., 2011. Workplace warriors: identifying team practices of appropriation in software ecosystems. In Proceeding of the 4th international workshop on Cooperative and human aspects of software engineering. CHASE '11. Waikiki, Honolulu, HI, USA: ACM, pp. 57–60. © 2011 ACM, Inc. http://doi.acm.org/10.1145/1984642.1984656
- **Chapter 6:** Draxler, S., Stevens, G., Boden, A., 2014. Keeping the development environment up to date - A Study of the Situated Practices of Appropriating the Eclipse IDE, IEEE Transactions on Software Engineering (TSE), accepted with minor revision. © 2014 IEEE. Reprinted, with permission, from Sebastian Draxler, Gunnar Stevens and Alexander Boden, September 2014. http://dx.doi.org/10.1109/TSE.2014.2354047
- **Chapter 7:** Draxler, S., Jung, A. & Stevens, G., 2011, End-User Development, Third International Symposium, IS-EUD 2011. Torre Canne, Italy, June 7-10, 2011, Proceedings, Managing software portfolios: A comparative Study. Springer, Berlin Heidelberg, pp. 337-42. With kind permission from Springer Science+Business Media. http://link.springer.com/chapter/10.1007%2F978-3-642-21530-8_36
- **Chapter 8:** Draxler, S., Sander, H., Jain, P., Jung, A. & Stevens, G., 2009. In Supplementary Proceedings of the 11th European Conference on Computer Supported Cooperative Work, Peerclipse: Tool Awareness in Local Communities. Vienna, Austria, pp. 19-20.
- **Chapter 9:** Draxler, S. et al., 2012. Supporting the social context of technology appropriation: on a synthesis of sharing tools and tool knowledge. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '12. New York, NY, USA: ACM, pp. 2835–2844. © 2012 ACM, Inc. http://dx.doi.org/10.1145/2207676.2208687

Additionally, these publications contribute to the presented topic. However, they are not included as sections of this book.

- Draxler, S., Sander, H. & Stevens, G., 2008, ECAI '08 Workshop on Recommender Systems, Plug-in recommending for Eclipse users. University of Patras, Patras, pp. 61-2.
- Pipek, V., Stevens, G., Veith, M., Müller, C. & Draxler, S., 2008, 16th European Conference on Information Systems, Towards an Appropriation Infrastructure: Supporting User Creativity in IT Adoption. Galway, Ireland, pp. 1165-77.
- Dörner, C. et al., 2009. End Users at the Bazaar: Designing Next-Generation Enterprise Resource Planning Systems. IEEE Software, 26(5), p.45–51.
- Draxler, S., Sander, H. & Stevens, G., 2010, Multikonferenz Wirtschaftsinformatik 2010, Provisioning 2.0: Diffusion kleinteiliger Software in sozialen Netzwerken. Universitätsverlag Göttingen, Göttingen, pp. 665-77.
- Stevens, G. & Draxler, S., 2010, Proceedings of the 9th International Conference on Designing Cooperative Systems, Appropriation of the Eclipse Ecosystem: Local Integration of Global Network Production. Springer, London, pp. 287-308.

Contents

> *"The third and final main area of impact on the software engineering practices for an organization transitioning from a software product line to a software ecosystem is the change in ownership of the product composition. [...] the immediate consequence of this approach is that the party composing the functionality of the overall solution is no longer the product line company, but instead the customer. The customer composes the solution that best suits his or her needs and assumes the resulting selection works seamlessly without requiring any additional work."* (Bosch 2009, pp. 8-9)

1. Introduction

The development, distribution and marketing of software products and services is continuously changing. The interplay of the latest major innovations over the last 20 years has created not only interesting possibilities for software producers, but also new challenges for its users and with that, new challenges for the fields of CSCW and HCI.

From a producer's perspective, the internet has become an infrastructure for multiple development and marketing activities. It is used to foster collaboration in several areas of product development, allowing for development methods as component software (Szyperski, Gruntz & Murer 2002) or Global Software Development (O'Brien & Montazemi 2003). However, it has also evolved as a vehicle for transporting and provisioning digital goods to customers through e.g. digital market places (Anderson 2009).

Accompanying these technical and work-practice related changes, producers have established new business models that actively encourage users to modify and share software (instead of simply sanctioning this behavior) (Dix 2007; Bourguin, Lewandowski & Lewkowicz 2013). Today, many producers clearly count user-developed contributions as part of their product development process, as part of their product experience and as part of the economic value chain (von Hippel 2005; Reichwald et al. 2004; McKinsey Global Institute et al. 2012).

In several cases of such kinds of flexible software, different software producers have even linked up to what can be described as a loosely coupled production network and which forms the base for software ecosystems (Messerschmitt & Szyperski 2003). These ecosystems often rely on a shared platform that defines a base and interfaces as well as certain standards, enabling the participating producers to contribute functioning additions. This allows manufacturers to deliver complex applications based on small software components, produced not only by themselves, but by several independent manufacturers (Bosch & Bosch-Sijtsema 2010a).

Described primarily from a production or economic point of view, these trends also have the potential to massively change the basic conditions for diffusion, usage and appropriation of software in recent years. Therefore, at this point I will change the perspective and discuss how these changes present themselves, from a user's point of view.

Not long ago, software was primarily delivered on storage media (such as floppy disks or CD-ROMs), combined with a printed manual. The internet as an infrastructure for transport, provisioning and marketing, led to digital market places where users obtain a large amount of their software today. Examples of this are Google Play and the Apple store for mobile products such as smartphones or Steam for computer games. In combination with the ecosystems trend, a growing number of software systems are based on (an existing and continuously growing number of) software components, created by different vendors (Bosch & Bosch-Sijtsema 2010a). This leads to a new freedom for users, as they can easily compose and modify systems and applications to their liking. But, this also implies that systems are, to some extent, put together by users themselves, e.g. by adding components. In the past, the producer was usually responsible for integrating components, as well as for testing the quality of the compositions. While the building blocks today are still delivered by professional producers, the integration work has shifted to become the users' responsibility. Naturally, producers try to ensure that integration (e.g. the installation of a smartphone app or an add-on for a web-browser) works as smoothly as possible. However, as small as this task may seem, it results in component combinations which may never have been tested before. Should incompatibilities arise, the user is left with a problem which generally belongs to the domain of software engineering work. License models which are free to use and adapt (e.g. Open Source Software) further encourage users to play around and explore, yet they also multiply the possible occurrence of such pitfalls.

1.1. A grounded motivation

It is highly likely that past and ongoing trends of software production ecosystems affect the usage, appropriation and acquisition of software from a user's point of view. Ecosystems and product communities empower users to explore different solutions and to exchange experiences with others throughout the whole world (Bourguin, Lewandowski & Lewkowicz 2013). At the same time, additional integration work might have negative effects on appropriation. The most influential studies on tailoring, appropriation and diffusion of complex software either predate (e.g. Mackay 1990b; Gantt & Nardi 1992; Kahler 1995; Ruel 2002; Trigg, Moran & Halasz 1987; Mørch 1997; Wulf & Golombek 2001) the introduction of software ecosystems (Messerschmitt & Szyperski 2003) and related developments, or focus on different aspects (e.g. Dourish 2003; Pipek 2005; Stevens 2009). Existing work therefore does not acknowledge the aspects of current production trends. To understand new opportunities for appropriation as well as new challenges caused by ecosystems and product communities, this line of work has to be complemented. I agree with Stevens, who argued in favor of the empirical investigation of *"self-organized communities (like talking about artifacts with friends, sharing adaptations with colleagues, etc.)"* (Stevens 2009) and for supporting the self-organized configuration management of software, surrounded by product communities (Stevens 2009). Within this work I accordingly intent to investigate how the current trends of software production are reflected in software appropriation in local (work-)groups. The main goals are:

- to gain a deeper understanding of activities contributing to appropriation, under the influence of modern software engineering trends (especially software ecosystems) and
- to assess possibilities for appropriation support under these circumstances.

In order to attain these objectives, this work is based on empirical field studies to investigate work groups, which are using the Eclipse Integrated Development Environment (IDE). Eclipse is a very flexible software system, developed by a globally active ecosystem of large corporations, small businesses and even hobbyists. Seen through the lens of appropriation (Dourish 2003; Pipek 2005; Stevens 2009; Dix 2007; Bourguin, Lewandowski & Lewkowicz 2013), practices such as learning how to use Eclipse in certain situations, sharing information and artifacts (e.g. components), modifying Eclipse and awareness about appropriation efforts are identified and discussed in the light of new production trends. Grounded in these results, suggestions for the design of appropriation support in the light of ecosystems are given.

1.2. Structure of this book

This book consists of four parts. Part II and III of this thesis have already been published in the form of several peer-reviewed publications, which constitute the core of this work. For a list of papers in chronological order see Related Publications, page V.

- **Part I** presents the background and frame for this work. It defines the necessary terminology and introduces existing work in the field of software appropriation and related topics. It furthermore frames this study by developing a research perspective, establishing research questions, presenting the way chosen to approach them and the selection of the field for empirical investigations.
- **Part II** presents those findings of my work that contribute to a) understanding activities which contribute towards appropriation by collaborating users under the influence of modern software engineering trends. This is presented as collection of peer-reviewed articles.
- **Part III** presents those findings that relate to b) the design for appropriation support, specifically to address some of the observed phenomena root in software ecosystems and product communities and which are new compared to other studies. This is presented as collection of peer-reviewed articles.
- **Part IV** summarizes the results and discusses the overall contributions reflected against current literature, pointing out the originality of this investigation. Furthermore, open questions and links to future research are shown.

I.

Overview

2. Related work

This section introduces scientific literature that grounds and defines the fields of software ecosystems and appropriation and forms the foundation for understanding these phenomena in the context of the Eclipse IDE. This topic is entwined with several disciplines. Software engineering discusses related topics mainly from a production perspective (see section 2.1). CSCW and HCI first introduced the user's perspective and started exploring what users make of given systems (see section 2.2). As these topics are interrelated, both perspectives are introduced. As part of the software engineering section, an overview on Software Ecosystems is given, since the term is used throughout this book and divers publications never allowed to explore it in detail. Section 2.3 connects both lines of research and argues for further research.

2.1. Components in modular Software Engineering

The discipline of software engineering defines the context for understanding the production side that influences Eclipse appropriation. One of the major problems of software production is that the fields of application are usually heterogeneous and change dynamically. This is especially true if software is created for socio-technical contexts (Orlikowski 1993; Wulf 1994; Wulf & Rohde 1995). In these contexts, end-users working and collaborating in organizations are involved. As organizations, tasks and relations between roles change, a perfect requirements analysis is impossible. Software engineering therefore has to cope with a continuous incompleteness of specifications (Firesmith 2005). This challenge often results in software that does not fit work practices well enough. Several possible procedures to mitigate effects have been established in the domain of software engineering. The following sections discuss software component approaches, as an architectural means to creating systems that can be more easily adapted to changing requirements than monolithic systems as well as the shift towards software ecosystems that is related to this technology. While the broader concept of tailorability also introduced approaches to cope with this problem, it stems more from a more direct work practice centered perspective of CSCW and HCI research and is therefore discussed in section 2.2.

2.1.1. Component approaches for modular software systems

Component approaches aim at the loose coupling of building blocks, which ideally can be replaced, tailored, inserted or removed from a software system during its runtime. Modern component-oriented systems additionally contain a container concept, which

implements a runtime environment for the components and delivers compositional tools.

The term *Component-oriented software construction* was first used by McIlroy (1968). His vision of future software production was that the work would be carried out mainly by combining ready-made building blocks. A more modern and detailed perspective on software components is given by Szyperski (2002). He points out that there is no single definition of the term software component, but argues for certain commonalities that components usually feature: *"A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Components can be deployed independently and are subject to composition by third parties."* (Szyperski, Gruntz & Murer 2002, p. 548)

Szyperski (2002) described component software as a valid medium between standard software and perfectly adapted, custom made software. He argues that it is easier to adapt to the users' needs than standard software. Yet at the same time, it is faster to develop (by using pre-existing components) and usually better tested (single components are well tested as they are used in multiple products) than custom made software. Overall, this allows better quality standards to be attained and also enables a faster reaction to the changing needs of users and organizations alike (Shaw & Clements 2006).

Over the years, a vast array of component approaches and de-facto standards has evolved. Szyperski (2002) separates these into three major categories: the Object Management Group (OMG) approach, the Sun/Java approach and Microsoft's approach. All of these define different styles of components, composition systems and runtimes. The Object Management Group (OMG) is a large standardization consortium. It operates on a non-profit base in order to achieve interoperability in an open market for objects. Its main achievement is the *Common Object Request Broker Architecture* (CORBA) to support the development of distributed applications in heterogeneous environments. The Sun/Java approach started out in the late 90s with the introduction of applets, small components that could be integrated into websites. Later, more evolved approaches were established. For example JavaBeans to create and distribute graphical user interface elements (Hamilton 1997). Enterprise Java Beans (EJB) to encapsulate and distribute business logic (Johnson 2003). JXTA to create and distribute services and resources within peer-to-peer networks (Gong 2001). The Microsoft approach was less an idea to standardize the industry and more a campaign to support the interoperability of their own applications. OLE/ActiveX created a way for users to exchange (copy and paste) data between the different Microsoft office applications (Chappell 1996). It was extended to encapsulate application elements into websites

(similar to Java Applets however, ActiveX was released earlier). Later approaches include COM+ and .NET CLR. Furthermore, there was great commitment towards creating service-oriented architectures (SOA), which can be seen as a component approach that relies on existing standard web architecture (application servers, web servers, HTTP and TCP/IP protocols). These are mostly used to separate split business logic into services and to make these services location-independent (Erl 2005).

One additional component standard as well as one implementation is to be noted, as it will feature rather prominently in this book: the OSGi standard (OSGi Alliance 2005). The OSGi alliance specified a platform-independent dynamic approach to creating component software. Like Sun's approaches, it relies on the Java programming language and Java's mechanism for creating building blocks – the jar container format that is also used for creating libraries. It focuses on stability and robustness, instead of fine-tuned composing techniques. As such, it packages services into building blocks; therefore OSGi components can't really be composed during runtime. It specifies a fine grained lifetime model and allows components to be replaced during runtime (OSGi Alliance 2005; Cummins & Ward 2013). Since Eclipse version 3.0, its components, so-called plug-ins are based on the OSGi standard. They offer services, resources and points for extension within the Eclipse runtime. In doing so, they encapsulate all existing Eclipse functionality, including business logic as well as graphical user interface elements (Gamma & Beck 2004). Eclipse components reintroduced component-level dependencies (OSGi specifies only service and package level dependencies).

2.1.2. Component approaches built for user customization

The introduced component standards, as well as their respective possibilities for composition and runtime environments, are still most useful from an engineering perspective as their composition is carried out mostly as part of designing component software. However, other approaches exist, which allow users to compose components themselves in order to create completely new application software (Wulf, Pipek & Won 2008; Mørch et al. 2004). The FreEvolve (Stiemerling 2000) approach is an example of this. It uses FlexiBeans, an extended version of JavaBeans which is combined with a box-and-wiring approach to allow users to compose new component software. The box-and-wiring approach is easy to use, as it follows a very visual approach that usually supports drag&drop functionality to add components and wires. Components are often visualized as boxes, connections between components as wires. Evolve (Stiemerling 2000) and SwapBox (Tan, Esfandiari & Pagurek 2001) are examples of such composition environments.

This kind of work was recognized and appreciated in the field of CSCW, where it contributed to the discussion around end users tailoring and customizing software on their own (Trigg & Bødker 1994). Later, approaches that put this kind of flexibility in the hands of the users (instead of the designers) were classified as End User Development (Lieberman et al. 2006), which is also mainly discussed within the CSCW community.

2.1.3. Components for manufacturing purposes: Software Product Lines

As we shown in the last section, some component approaches exist that inherently allow users to compose systems. However, many approaches in use still mainly follow the needs of efficient software manufacturing. The software product lines approach (Clements & Northrop 2002) continues this tradition. Bosch's (2009) work in particular traces the development from component approaches towards Software Product Lines (and further towards Software Ecosystems). Software Product Lines can be seen as an engineering perspective on the development of extendable software platforms in order to create and deliver different versions of software products. To achieve this, the manufacturer splits a product into components and composes sets of components to deliver certain functionalities as a version of the product (Clements & Northrop 2002; Bosch 2009). This results in specific base building blocks having to be developed and tested only once for all products, thus rendering different versions of a software product cheaper to develop (Northrop & Clements 2004). Northrop and Clements (2004) defined the technological base for software product lines as:

- A commonly used platform that is used to integrate all the building blocks.
- A component approach that is used to define the building blocks of user functionality or basics as a UI framework.
- Clearly defined interfaces between components, as well as between components and the platform.
- A common release cycle in order to synchronize the development of platform and components.

This new approach was initially seen as an interesting new production technique for large manufacturers who were able to develop platforms as well as the extensions on their own (Northrop & Clements 2004). This implies that the integration work necessary to ensure that components form a valid, executable product is carried out by the manufacturer (Bosch 2009). This viewpoint changed with the advent of software

ecosystems, as platforms as well as development processes were opened for other manufacturers to join in (Bosch 2009).

2.1.4. Software Ecosystems

The first usage of the term Software Ecosystem is assigned to Messerschmitt and Szyperski (2003), who used it to introduce a new perspective on software engineering to the scientific discussion. Since then, the term has undergone a number of reinterpretations. This section gives an overview of the different lines of work in this area and defines how the term *Software Ecosystem* is used within this book.

Messerschmitt and Szyperski (2005) argued (and different researchers seem to agree here (Bosch & Bosch-Sijtsema 2010b; Jansen, Finkelstein & Brinkkemper 2009)), that the market and the conditions for software production and distribution changed over time. They observed that production became more fine grained and involved more and more developing actors. Such actors can be individual persons as well as organizations. Furthermore, they agree that those actors nowadays often group around certain commonly used platforms. A typical example is the software manufacturer that opens up the base or platform of a software to other actors and allows them to develop additional components that deliver additional functionalities for the end-user. Bosch and Bosch-Sijtsema defined this as follows: *"A software ecosystem consists of a software platform, a set of internal and external developers and a community of domain experts in service to a community of users that compose relevant solution elements to satisfy their needs." (Bosch & Bosch-Sijtsema 2010b)*

Jansen et al. (2008) in contrast, focus even more on the software engineering or manufacturing side as they emphasize the connections between the involved developing actors: *"We define a software ecosystem as a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts." (Jansen, Finkelstein & Brinkkemper 2009)*

Messerschmitt and Szyperski's work (2005) is in line with these definitions. However, it additionally created a new perspective on the development and production of software. Their ecosystems perspective focuses on all typically involved stakeholders and their share of the process. This allows involved actors to understand the problems, needs, skills and potential of other stakeholders, possibly leading to a more refined understanding of their own work. On the one hand, this can be seen as practical use for the involved agents, but it is more than just that. It is a manifesto for a multi-per-

spective view on the development, marketing and distribution of software – which, as the authors argue, is increasingly gaining in importance in the struggle to remain successful in the software business of tomorrow. In the main however, it is this multi-perspective view in particular which is missing from the work of others (Bosch 2009; Jansen, Cusumano & Brinkkemper 2013; Joshua et al. 2013), as they argue only from a manufacturers point of view.

One of the important changes accompanying the introduction of software ecosystems is that platforms (and parts of the common process) are opened up for others to participate. When this happens, a plethora of actors can develop and publish their own components and, as a result, the responsibilities of the actors change. Jansen et al. (2009) described the layers of responsibility that are usually established after opening up processes in this way: *"software ecosystem level, the software supply network level, and the software vendor level." (Jansen, Finkelstein & Brinkkemper 2009)*

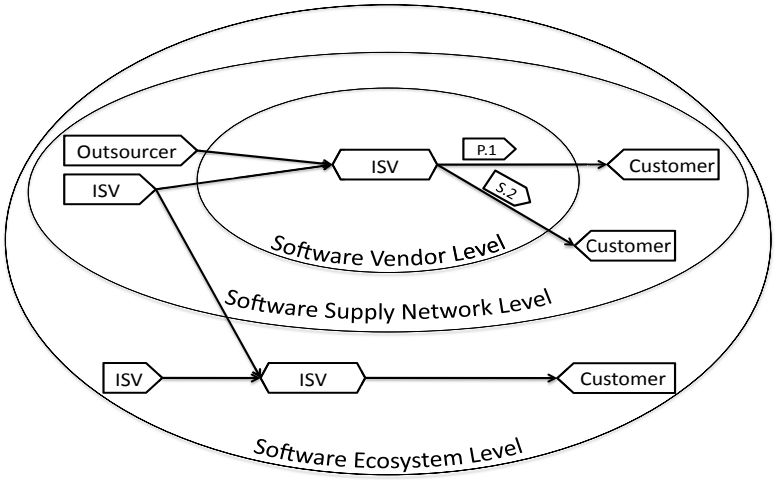

Figure 1: Software Ecosystems Perspectives (Jansen et al. 2009).

As the platform owner's responsibility for integrating and testing possible compositions lessens, some of the integration work is shifted to the customer:

"The third and final main area of impact on the software engineering practices for an organization transitioning from a software product line to a software ecosystem is the change in ownership of the product composition. A software ecosystem consists of a platform, products built on top of that platform and applications built on top of the platform that extend the products with functionality developed by external developers. However, the immediate consequence of this approach is that the party composing the functionality of the overall solution is no longer the product line company, but instead the customer. The customer composes the solution that best suits his or her needs and

assumes the resulting selection works seamlessly without requiring any additional work." (Bosch 2009, pp. 8-9)

In many cases however, the customer may also be the user. Therefore in certain situations, the integration work such as composition and/or testing is outsourced to the end-user. Examples span from Web browsers like Mozilla Firefox through Microsofts Office suite and games to CAD software and Integrated Development Environments (see section 3.3 for a detailed analysis of examples).

2.1.4.1. A classification of Software Ecosystems

Bosch (2009) separates existing software ecosystems along two dimensions: Different kinds of software (end-user programming, application or operating systems) and the underlying platform-technology (desktop, web or mobile). Table 1 gives a brief overview of several modern software ecosystems, classified by this scheme.

Category\Platform	Desktop	Web	Mobile
end-user programming	MS Excel, Mathematica, VHDL	Microsoft PopFly, Google's mashup editor	Tasker (Android)
application	MS Office, Eclipse, Firefox, Photoshop, Skype, WoW	SalesForce, eBay, Amazon, Ning	Yatse (Android)
operating system	MS Windows, Linux, Apple OS X	Google AppEngine, Yahoo developer, Cloghead, Bungee Labs	Nokia S60, Palm, Android, iOS

Table 1: Classification of software ecosystems, following (Bosch 2009) and (Joshua 2013), extended and updated.

Operating systems count as the oldest software ecosystems, as they are by nature open to the extensions of other software manufacturers (Bosch 2009). Manufacturers of operating systems have always delivered a profusion of documentation for the development of additional building blocks such as drivers for hardware or application software for end users. For Bosch, the new development within the software engineering community is that manufacturers have intensified their commitment to do the same in the area of application software.

Application-centric software ecosystems are set up to allow independent software vendors, distribution partners, consulting and customization organizations to create and distribute their own extensions and services that integrate seamlessly into the application. Joshua et al. (2013) picked the software development environment Eclipse as

an example. But other integrated development environments like Netbeans, Visual Studio or IntelliJ IDEA seem to aim in the same direction.

Furthermore, web applications such as Facebook feature possibilities to embed additional applications into these services (Graham 2012). Current examples mainly include games, but also include music streaming or news services[1].

Bosch only named operating systems within the mobile category. However, this can be complemented by apps such as Yatse[2] and Tasker[3]. Yatse is a mobile app for the Android OS that controls media center software by remote. It is extendable through several small components. Tasker is also an Android application which can be classified under End User Development. It allows users to automate different tasks, based on different contexts and sensor values of the device. E.g. if the user is within the range of a particular wifi network, the context *home* might be activated. The device might then activate certain tasks, as configured by the user.

Bosch (2009) assessed end-user programming ecosystems as not being of very great importance, although he appreciated their special characteristics. He asserts that they are powerful enough to let users create their own software without greatly increasing flexibility. For a long time, it was believed that this outstanding property would one day let users widely create their own software (McIlroy 1968). Many of the existing tailoring approaches are grounded in ideas such as this (Wulf, Pipek & Won 2008; Mørch et al. 2004). Today there are several commercial approaches in existence, such as e.g. Yahoo Pipes[4]. Research, however, is still ongoing, using the terms End-User Development (Lieberman et al. 2006; Pipek et al. 2009; Costabile et al. 2011; Dittrich et al. 2013) and End-User Software Engineering (Ko et al. 2011; Burnett, Cook & Rothermel 2004; Burnett 2009). The scientific discourse mainly covers desktop applications (Le Berre & Rapicault 2009; Bosch 2009). However, web applications (Jansen, Finkelstein & Brinkkemper 2009) and embedded systems (Bosch & Bosch-Sijtsema 2010b) have also been explored.

1. https://www.facebook.com/appcenter/category/apps/?platform=web (last accessed 2014/04/01)

2. http://yatse.leetzone.org/redmine/projects/androidwidget/wiki (last accessed 2014/04/01)

3. http://tasker.dinglisch.net/ (last accessed 2014/05/15)

4. http://pipes.yahoo.com/pipes/ (last accessed 2014/04/01)

2.1.4.2. Eclipse as a software ecosystem

From an engineering perspective, the Eclipse IDE is a very good example of a software ecosystem. Using Bosch's (2009) classification system, it is an application, but at this point I would like to classify this in more detail. This will be extended in several sections of parts II and III.

The Eclipse IDE makes use of a platform, often referred to as the Eclipse platform or Eclipse Rich Client Platform (RCP). The platform as well as the first IDE functionalities were implemented by IBM and later passed into the hands of a committee (the Eclipse Foundation) (O'Mahony, Diaz & Mamas 2005). Furthermore there is a development process in place, which is coordinated via forums, interfaces between components and the platform and a certain development rhythm (Gamma & Beck 2004; Frost 2007). The development efforts of certain organizations are interlocked with the work of the Eclipse foundation and other organizations. However, other manufacturers are completely independent and contribute to their own liking and at their own pace.

It is noteworthy that the development of the platform as well as the IDE functionality is carried out by this plethora of organizations. In comparison the existing software ecosystem definitions, this is a special case as no single manufacturer is responsible for the platform. This complicates the development of the Eclipse IDE. Some of the contributing organizations are, to some extent interconnected due to technical dependencies between their products. This usually happens if one manufacturer's components make use of another manufacturer's components. As the interface of one component evolves within a new Eclipse release, other components may also need changes that lie within the responsibilities of another manufacturer (Des Rivières & Wiegand 2004).

There is a market for components that are compatible with the Eclipse platform. However, it is very distributed because there is no mandatory central coordination. Although there is a centralized plug-in directory (called Eclipse Marketplace[5]), it often contains outdated information. Some manufacturers don't even bother to register their components. Bosch's (2009) argument presenting the user as the compositor can be taken literally within this context.

5. http://marketplace.eclipse.org/ (last accessed 2014/05/14)

2.1.5. Modularity in Integrated Development Environments

The Eclipse IDE, as the main field of research in this book, shows that workplaces for software engineering usually combine plenty of different tools and integrate their features into a powerful and often complicated workplace (Des Rivières & Wiegand 2004).

However, flexible software systems (IDE are a particularly good example) have come a long way. Early programming tools were small and simple. They were specialized for the automation of routinized, basic tasks (e.g. compilers to generate machine code) and it was not uncommon to create one's own tools (Kernighan & Plauger 1976). In the 1970s however, the complexity of the supported tasks was increasing (Bennett et al. 2008) and the variety and number of tools was constantly growing (Forte 1993).

Tools were mostly developed independently from each other. While each task was improved singly, it was often clumsy or not possible at all to use the various tools together. As a result, research shifted during the 80s from the isolated tool perspective to integrating tools into supporting environments (Forte 1993; Boehm 2006). In the following, this shift is outlined along general lines.

Emacs can be considered one of the first and notable answers to the challenge of tool integration. From the beginning, Stallmann (1981) aimed at creating a powerful text editor that could be tailored and extended by the users *"to experiment with alternative command languages, and to share extensions which are generally useful"* (Stallman 1981). Examples for extensions (e.g. Glickstein 2010) that add new functionality to Emacs are object oriented programming (Houser & Kalter 1992) or prolog programming (Bueno et al. 1997).

Significant efforts supporting the entire development process beyond individual tasks were carried out within the field of Computer Aided Software Engineering (CASE). CASE had also emerged in the 1980s (Chikofsky 1989). With increasing popularity, CASE became an umbrella term for diverse SE concepts including tool integration, automated tool assistance, process standardization etc. (Fuggetta 1993). The common idea was to address *"all aspects of software development and maintenance as a single complex system"* (Forte 1993), including modeling, programming, debugging, managing requirements, documenting, creating user interfaces, etc. One unexpected result of the CASE approach was that the individual developers were mainly excluded from the design of their own workplaces. Only the freedom to change small details such as shortcut keys or reorganizing windows remained their responsibility.

The latest pragmatic turn can be described as distinguishing repetitive software engineering tasks that can be streamlined via automation from the "essential" tasks that rest on human expertise, judgment, and collaboration (Brooks & Jr. 1987). This turn is expressed by less restrictive integrated development environments. In particular IDEs like Microsoft's Visual Studio and Eclipse as two widespread examples (Geer 2005) present a synthesis of the need for freedom and the need for standardization: On the one hand they maintain the concept of freedom, which allows users to adapt the working environment to their needs without over-regulating the situated work (Gamma & Beck 2004). On the other hand they contribute to the automation of repetitive tasks as well as standardization. Today, such tools are usually extensible by using various components that add more functionality as well as including tailoring functions (e.g. windows, fonts, font sizes, style of compiler, etc.).

During the 1990s the standardization of working environments had also progressed, e.g. by the introduction of the NIST/ECMA (1993) reference model for tool integration. However, despite that progress, working environments today often consist of many tools which are only *"partially integrated, forming a complex tool landscape with partial integration."* (Asplund et al. 2011). One of the reasons for this ongoing struggle is that not only the solutions, but also the industry's demands are constantly evolving. For instance, new methodologies like aspect-oriented programming, model driven development or continuous integration emerged. In addition, new application areas like mobile computing, web applications or the internet of things arose and demanded special tools.

2.2. Situated use and appropriation of modular software systems

This section introduces the concept of customization of flexible software by its users. The overview is guided by our primary research interest: appropriation of flexible software (as part of workplace design) in the age of software ecosystems. This kind of research was mostly conducted within the CSCW research community. As this topic is the core of this book, several aspects are introduced later in minute detail (see e.g. section 4). Here, an overview is given.

The presented work differs from section 2.1 mainly in perspective. While section 2.1 is driven by a manufacturing perspective, the following subsections take a CSCW perspective. Common to many CSCW studies is the focus on the user, taking the embeddedness of usage and customization activities into organizational and daily routine very seriously (c.f. Bentley et al. 1992; Orlikowski 1992).

The existing work concerning flexible systems from a user perspective, can be differentiated into two groups: first generation studies that concentrated on the act of tailor-

ing and customization and second generation studies that used a much broader understanding of the topic at hand.

2.2.1. The 1st generation of studies on flexible system usage: tailoring

Efforts to create tailorable or customizable software predate Orlikowski's (1993) seminal empirical field study by far (i.a. Stallman 1981). However, her results do explain very illustratively why this development is so important. Software engineering is often not able to keep up with the speed at which, organizations and their requirements were changing (Kahler 1995). During this time, communities like CSCW and HCI started to cultivate interest in flexible software concepts, analyzing and extending the technological state, but also investigating how users cope with flexible software systems.

Early important examples of such work are (i.a. Henderson & Kyng 1992; Wulf 1994; Muller, Haslwanter & Dayton 1997). In this line of work, different tailorable systems were created as research prototypes to explore the possibilities of tailorable software. Examples are DODE (Fischer 1994), OVAL (Malone, Lai & Fry 1995), Prospero (Dourish 1996), or FreEvolve (Wulf, Pipek & Won 2008). Next to concepts pertaining to how tailorability should be established from a software engineering point of view, they also helped to gain a better understanding of different levels of tailoring complexity (and possibilities) (MacLean et al. 1990). E.g. changing the font size of an application might be easier than programming a macro for a text processor. At the same time, programming might allow greater possibilities than functionalities which are easier to use. As a result, those studies demanded a so called *"gentle slope of complexity"* (MacLean et al. 1990). By recognizing this, they illustrated that users have different motivations towards, as well as skills in, tailoring software. people need a smooth transition from a tailoring mechanism which is easy to use but not powerful to a more powerful but also more complex one, especially if they are not skilled in formal programming (which is true for most end users). Mørch (1997) investigated such levels of tailorability in detail and introduced more refined borders between different tailoring levels. He distinguishes customizing, integration (e.g. composition of components) and extending (e.g. programming a macro) (Mørch 1997).

Some viewed tailoring mainly as an individual effort (i.a. Kobsa & Wahlster 1989; Oppermann 1994; cf. Friedrich, & Rödiger 1991). Examples are modifying toolbars in a text processor or the font size of the operating system. This perspective also implicitly includes the possibility that one person's requirements may differ considerably from another's thus necessitating this depth of flexibility. However, as empirical studies with users of commercial systems in the wild have shown, there was substantial inter-

est in the tailoring results achieved by others within collaborative environments (i.a. Mackay 1990a; MacLean et al. 1990; Gantt & Nardi 1992; Wulf 1999b; Kahler 2001b; Eriksson 2008). This established the perspective that tailoring is a collaborative effort. This perspective gave rise to interesting new aspects, such as *who tailors?* (capabilities/skills of users) and *in what ways are results shared?* (division of labor), etc. As a result, different studies established classification systems for users and their abilities to tailor (Gantt & Nardi 1992; Mackay 1990b) as well as sharing habits (Mackay 1990a).

In the late 1990s, a range of studies was conducted in the intersection between CSCW and HCI which demanded additional supportive technical functions to make it easier to cope with tailorable systems. Examples are the concept of *direct activation*, which demands that the means to tailor a function should be coupled to the point of entry of that function (Wulf 1999a). Kahler (2001b) furthermore demanded support such as repositories for tailored artifacts to choose from, as well as mechanisms for sharing and awareness support. However, Kahler did not specify concepts for the realization of such features.

2.2.2. The 2nd generation of studies on flexible system usage: appropriation

The work on tailorability has mostly been succeeded by other lines of research today. The technical aspects of empowering (end-)users to make the most of flexible systems has been subject of discussion for at least 10 years within the End User Development (EUD) community, which biannually meets to present the theoretical and technological developments as well as studies concerning EUD systems (Lieberman et al. 2006; Pipek et al. 2009; Costabile et al. 2011; Dittrich et al. 2013). Analytically however, the research around appropriation of software systems can be seen as a successor of previous tailorability research.

Appropriation is considered a much broader term, that cannot be reduced to pure tailoring or customization of software. Instead, it is the general prerequisite of making practical use of (information) technology (in this case software). Part of this can be the repurposing of artifacts or their modification. A distinction between activities of plain usage and activities that accommodate appropriation are often not separable at all (Pipek 2005; Dourish 2003; Balka & Wagner 2006; Stevens 2009).

Historically, the concept stems from the tradition of information systems (IS) and was introduced to help information system models to include the rather vaguely definable results of system usage: *"we cannot expect people to employ a technology in predefined ways. Rather, we can expect that users will alter a system as they use it. The key is to understand how users alter systems, thereby enacting sociotechnical change within the*

group. Adaptive Structuration Theory defines this process as 'appropriation.'" (Poole & DeSanctis 1989)

Dourish (2003) carefully introduced the concept into the CSCW discourse by using *"the term 'appropriation' to refer to the ways in which people adopt and adapt interactive technologies, fitting them into working practices and evolving those practices around them."* (Dourish 2003) This perspective intensified pressure to think of gaining and sharing knowledge about systems or repurposing artifacts without tailoring. Still, tailoring is an aspect of appropriation.

Based upon this perspective, several studies have been carried out. Pipek (2005) carried out two long term studies, observing appropriation activities (one in a German authority and one within a commercial network of consultants. Other studies investigated appropriation of technology by teenagers (Carroll et al. 2002), in hospitals (Balka & Wagner 2006), photo logs (Khalid & Dix 2010) and software development (Sigfridsson 2010).

Conceptual advances of appropriation research include classification schemes for tailoring activities as part of appropriation (Kahler 2001b; Pipek 2005) and classification schemes for the dimensions and interplay of appropriation with the surrounding situation (Balka & Wagner 2006). Stevens (2009) opened up these results and created an extensive theoretical investigation that put appropriation back into the context of software development and connected it with ideas of open innovation and global appropriation infrastructures, after the general idea was described by Pipek (2005). This perspective regarding the way a user's needs can be combined with new business models and flexible systems is one of the reasons for this book. Twidale (2005) established the concept of informal, spontaneous workplace help-giving among colleagues that is very closely connected to the appropriation of software. Additionally, a bird's eye view model of technology appropriation was suggested (Carroll 2004). This, however, focuses more on the individual and therefore ignores many results from the CSCW community. Furthermore, the importance of breakdown situations for appropriation was discussed (Pipek 2005; Stevens, Pipek & Wulf 2009).

Several researchers argued in favor of additional (or differently designed) technology to support appropriation. Pipek (2005) argued for supporting appropriation by empowering users to take part in design discourses and to share experiences. He suggested a technological infrastructure to support this. Stevens discussed these ideas for support e.g. use-development discourses in more detail and argued in favor of integrating these directly into the application. Bourguin, Lewandowski & Lewkowicz (2013) used the prototype ShareXP to argue for and demonstrate a similar approach, focusing on sharing of experiences and related tailoring artifacts. Concerning several details of the

ShareXP concept as well as the field of application, this work is very related to what is presented here. A more general approach, stemming more directly from the field of human computer interaction suggests certain aspects that the design of software should include to accommodate appropriation (Dix 2007). Examples of this are flexibility (e.g. component software), allowing for reinterpretation (not too specifically designed functions) and enabling users to share artifacts.

2.3. Appropriation of software ecosystems

From a software engineering perspective, appropriation is not without problems. Users changing their own work practices as part of appropriation can lead to new requirements, which can render existing products to be less useful and in need of additional development work (Firesmith 2005). As demonstrated, the technology to create flexible software that could be leveraged by users for modification mostly already exists (i.a. Wulf, Pipek & Won 2008; Mørch et al. 2004). However, component approaches are still often used in ways which benefit the manufacturer, not the user. The software product line trend is one example of this (i.a. Clements & Northrop 2002).

The trend of software ecosystems is however different. The structure of collaboration between the participating actors and market places (Messerschmitt & Szyperski 2005) demands models of flexibility that users can interact with. Detailed examples for this development are shown in section 3.3 and include web browsers, games, word processors, CAD software, mobile devices and Integrated Development Environments (IDE).

How appropriation of software that was produced by such complex, distributed and collaborative environments is constituted – is largely unknown. For an investigation into this matter, the concept of appropriation seems an adequate lens to uncover specific aspects about using, reinterpreting, maintaining and modifying ecosystem-produced software.

3. Research Outline

The following section will elaborate on the research perspective of this work and carve out the research questions towards which the related work is leading. Accordingly, the methodology to approach these questions is discussed, ending with a grounded description of the methods used for the case studies as well as the design efforts.

3.1. Research Perspective

Based on the body of research concerning the development and appropriation of flexible software systems (see section 2), it can be concluded that software engineering of flexible systems is changing. Furthermore, while there is a solid body of work concerning tailoring and appropriation of flexible software in general, we do not know how the current software engineering trend of software ecosystems is affecting the usage and appropriation of such systems. This is especially remarkable as software ecosystems (e.g. users integrating software themselves, the plethora of different vendors involved, different release cycles of components) seem to possess special potential to change patterns of appropriation. A more profound investigation into the appropriation of software produced by a software ecosystem is therefore needed.

However, the use of information technology is embedded in personal preferences, social and socio-technical systems as well as in the product communities of software manufacturers (Poole & DeSanctis 1989; Dourish 2003; Wulf & Rohde 1995; Pipek 2005; Stevens, Pipek & Wulf 2010). Appropriation can therefore only be understood if these aspects are taken into account. In the past, the micro-perspective research approach of CSCW studies has proven to be very successful in achieving this (e.g. Heath, Luff & Cambridge 1992; Randall, Harper & Rouncefield 2005; Wulf et al. 2011; Eveland et al. 1994; Bowers 1994). This micro-perspective allows the focus of this investigation to fall on the main actor – the user. Starting from this point, social, technical, socio-technical and other influences can be studied in detail, while always allowing to view and hopefully understand their entanglements with the user. This work will follow this principle in order to uncover details of appropriation of flexible software, produced by a software ecosystem. In this way, I hope to gain a deeper understanding of what the global phenomenon of software ecosystems means for today's software users and which possibilities for support exist.

The term appropriation itself (Dourish 2003; Pipek 2005; Stevens 2009), introduced a very interesting lens to focus on phenomena related to usage, but often cannot be clearly classified as plain usage. Often, appropriation is expressed by work that is carried out in order to be able to use software or improve the usage experience, similar to

articulation work being defined as work *"that is necessary to be able to work"* (Strauss 1988). While appropriation is a very good guiding lens, it is still not defined clearly enough to classify what can be discovered through field work. It will therefore be used to guide the empirical view towards the user's practices but not to classify these in advance.

3.2. Research Questions

The research perspective underlying this work argued in favor of a closer look at the practices of users, trying to understand appropriation form this perspective. Against this backdrop, the following questions will be investigated and discussed within this work:

Q1) The constitution of appropriation: How do users, as part of work groups, appropriate flexible software that was produced and is surrounded by a software ecosystem? (What are the general conditions, which practices are applied to cope with such software and which situations contextualize appropriation?)

The explicit naming of work groups was chosen, as this opens up opportunities for observing possible practices of sharing expertise (Twidale 2005) or artifacts (Mackay 1990a) as part of appropriation. Furthermore it allows the investigation of such practices, with regard to the different existing organizational levels.

To investigate this question, different software systems are suitable fields of research. An overview as well as the justification for the choice that was made (the Eclipse IDE) is presented in section 3.3. Within this field of research, practices relating both to appropriation as well as to their contextualizing situations will be uncovered. Especially those that can be traced back to software ecosystems as a transforming factor will be discussed later in this book. This should not produce a normative description of good or bad practices, but a thick description (Geertz 1973) and analysis of the actual state of appropriation on the shop floor to broaden our understanding of appropriation.

Q2) Transferability: Are the results wholly or partially transferable to other software systems? / Can we find similar practices and phenomena in other software ecosystems?

Qualitative empirical studies stemming from the tradition of CSCW research usually produce theories of low range. Their results are therefore very detailed, however not easily transferable to other cases or even domains (Strauss & Corbin 1997). It is safe to assume, that this study is no exception, due to the very specific focus and applied methods. That said, as the software ecosystems trend is spreading, it would be very

interesting to investigate appropriation of other flexible software systems to compare results. As part of this study, it was possible to carry out an additional, albeit smaller, empirical field study in a different field of research (World of Warcraft).

Q3) Support: How is appropriation supported/accommodated by the software system? Can we observe problems and areas for improvement? How should adequate appropriation support be shaped?

An increasing array of software is currently equipped with features which allow the user to modify the system (see section 3.3 for several examples of component-based extendability). Sometimes complex and confusing product communities evolve around these features (Murphy-Hill & Murphy 2011; Stevens 2009). While some manufacturers are beginning to understand that flexibility is an interesting asset to their software systems, the software ecosystem approach seems to remove the clear responsibility of, who should manage those communities and take care of integration work (Bosch 2009).

As part of the empirical research in order to discuss Q1 and Q2, this work therefore also intends to inform design-activities carried out to discuss new appropriation support that might be beneficial, compared to existing approaches.

3.3. Selecting a field for empirical appropriation research

As software usage is embedded in and entangled with social, technical and often socio-technical systems (Poole & DeSanctis 1989; Dourish 2003; Wulf & Rohde 1995; Pipek 2005; Stevens, Pipek & Wulf 2010), it depends on both the private and work contexts of users as well as on the technological frame. The latter is continuously being developed further and this moreover potentially changes from one software system to the next. An empirical field study to discuss *Q1-Q3* universally is therefore not possible. Instead, this study is based on one exemplarily chosen field and opens possibilities for others to investigate furthers fields in order to compare and complement the results. At the time of this study, there were several very interesting examples of flexible software systems. Due to the focus on the appropriation of software systems that are part of a software ecosystem, the field of investigation needs to reflect this. The following line-up of software systems (see Table 2) is therefore fitted with clear features of tailorability and created by a software ecosystem.

	Type	Target audience	Tailorability	Community	Collaboration
Mozilla Firefox	Web Browser	not specialized	Look of the UI, UI Elements, Add-ons	centrally managed	Mozilla community
Adobe Photoshop	Graphics processor	Media Designers, Photographers	Plug-ins, actions	centrally + decentralized	unknown
Microsoft Office	Text processor	not specialized	Add-ins, User interface, Macros	centrally + decentralized	unknown; personal exchange
Google Sketchup	3D modeling	Architects, 3D design	Plug-ins	decentralized	forums
Eclipse IDE	Software engineering	Software developers, software testers, software architects, etc.	Plug-ins, preference settings, user interface	decentralized	forums; marketplace; contact developers
World of Warcraft	Massive multiplayer online game	Gaming	Add-ons, User interface	centrally + decentralized	local groups, curse community

Table 2: Characterization of different flexible software systems that are incorporated in a software ecosystem.

In the following sections, each software system from Table 2 is briefly analyzed to asses in which ways they are flexible and to what extent they are part of a software ecosystem. This is followed by a grounded selection of one software system as the field of research for this work.

3.3.1. Firefox web browser

Firefox is a general purpose web browser, used to surf the World Wide Web in order to do everything from gathering information through online shopping to hearing music and watching videos. Web browsers usually have no particular target audience and Firefox is no exception.

Firefox is developed by the Mozilla foundation, which describes the process as follows: *"The Mozilla project is governed by a virtual management team made up of experts from various parts of the community. Some people with leadership roles are employed to work on Mozilla and others are not. Leadership roles are granted based on how active an individual is within the community as well as the quality and nature of his or*

her contributions. This meritocracy is a resilient and effective way to guide our global community. [6]

Firefox uses so-called personas to change its visual appearance. User interface elements such as buttons for certain functions can be hidden/shown and reordered within a toolbar. Furthermore, a wide variety of add-ons exist that can provide powerful additional features. Examples are a proxy for anonymous surfing, download helpers for web content or a literature manager.

The Mozilla foundation provides a website, dedicated to different forms of add-ons (see Figure 2), that is also accessible as a native application within Firefox. It includes the sections *extensions* – small components that deliver new functionality, and *themes* – to change the optical appearance.

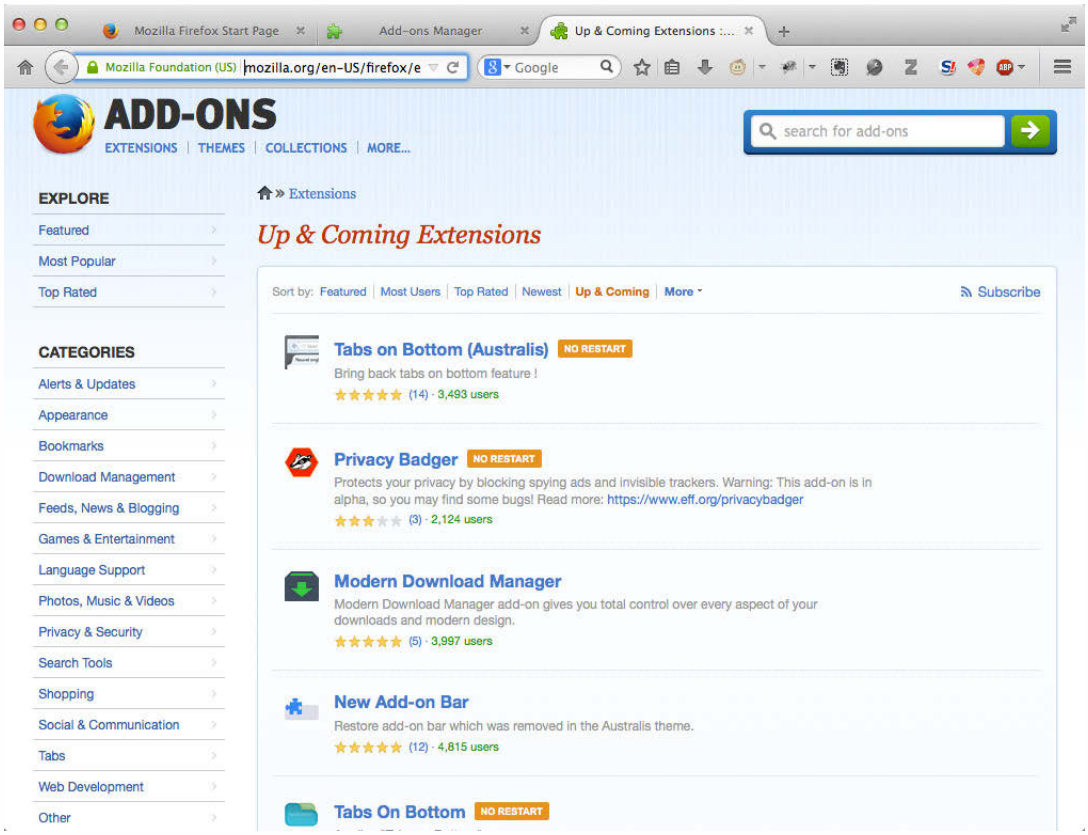

Figure 2: Overview of Mozilla Firefox add-ons. (https://addons.mozilla.org/en-US/firefox/extensions/ accessed: 2014/05/16).

Furthermore, there are community features such as most-popular add-ons, featured extensions, User-based rankings, and search for add-ons. Furthermore and maybe most interesting, a feature that allows sets of popular add-ons, which can be defined by oth-

6. http://www.mozilla.org/en-US/about/governance/roles/ (last accessed 2013/12/18)

ers and contain multiple extensions, to be installed. The Firefox add-ons site contains 139 extensions that deliver new functionalities[7].

3.3.2. Photoshop image processor

Adobe Photoshop is a very powerful image editor/graphics processor that over the years has become virtually an industrial standard.

The Photoshop user interface can be tailored by changing menu and palette entries. Palettes can be freely moved around. Furthermore, Photoshop has a long tradition of being extendable through plug-ins. Adobe itself provides a number of functions as plug-ins (e.g. the import of proprietary raw file formats of photo cameras); yet many plug-ins are created from within the Photoshop community. Plug-ins usually provide tools for filtering, importing or exporting, color correction or automation (Dayley 2012).

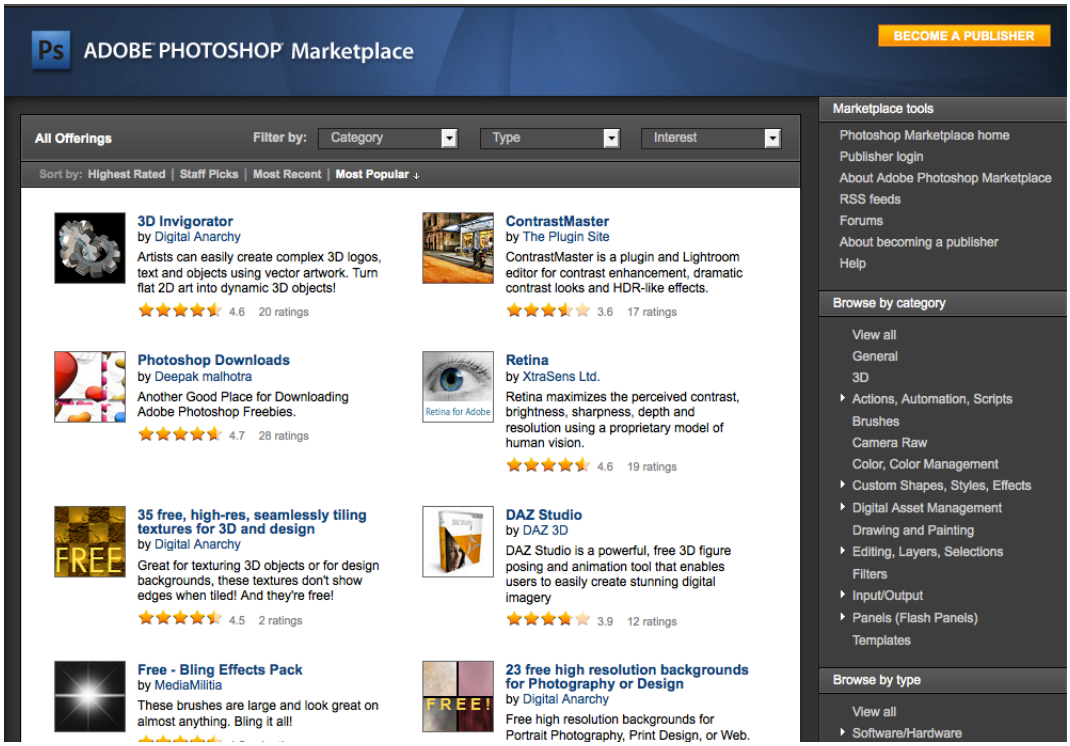

Figure 3: The Adobe Photoshop Marketplace, showing plug-ins for Photoshop (left) and categorization mechanisms for search functionality (right). (http://www.adobe.com/cfusion/marketplace/index.cfm?event=marketplace.categories&marketplaceId=2&categoryid= accessed 2014/05/16)

Adobe provides a community for plug-ins (see Figure 3), which groups plug-ins into categories and provides search a functionality. Several commercial companies have

7. https://addons.mozilla.org/en-US/firefox/ (last accessed 2014/05/15)

been manufacturing plug-ins for years. However, many of these provide their own communities with forums and market places or simple download possibilities (Dayley 2012).

At the time of this investigation, the Photoshop Marketplace featured around 1,897 plugins[8] delivering additional functionality.

3.3.3. Microsoft Office

The Microsoft Office suite features a general purpose word processor, spread sheets, and presentation software. Office is used by everyone from pupils to managers in order to create written documents, calculate costs or create presentations (Melton 2013).

Office has a long tradition of being customizable in many ways. The user interface (menus and toolbars) can be fully adapted. Repetitive tasks can be reproduced in so-called macros, small programs that are mostly generated by recording user interaction (Kahler 2001a). Furthermore, Office (especially the text processor Word) can be tailored by using add-ins, now called Office Apps (see Figure 4).

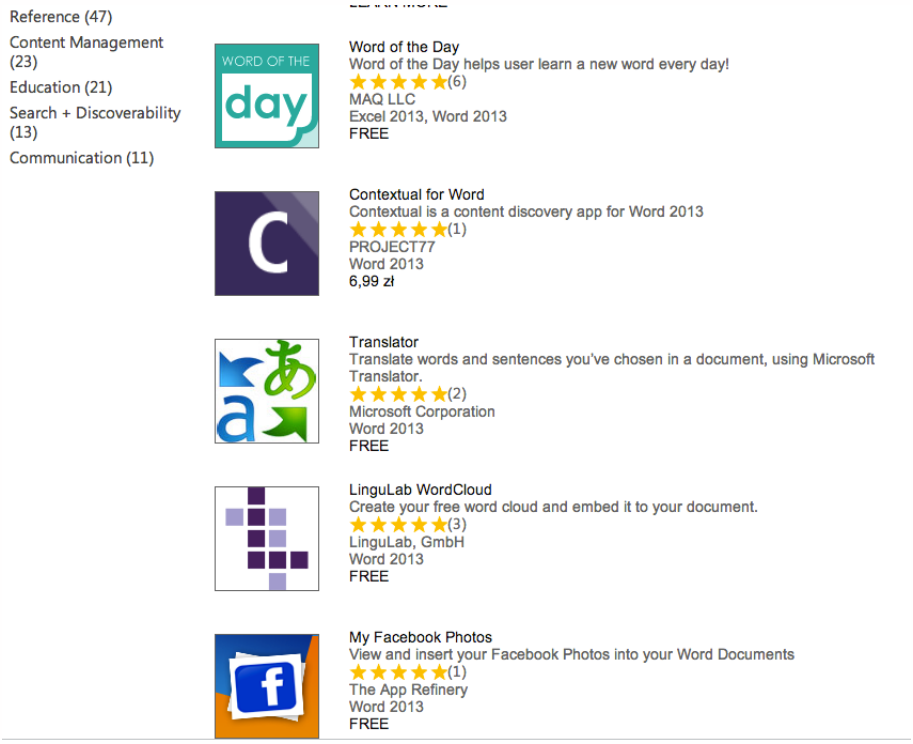

Figure 4: An overview of components in the Microsoft Store for office apps. (http://office.microsoft.com/en-us/store/?CTT=97 accessed 2014/05/16)

8. http://www.adobe.com/cfusion/marketplace/index.cfm?event=marketplace.categories&marketplaceId=2&categoryid= (last accessed 2014/05/16)

Existing add-ins for Word include dictionaries (Bing dictionary), encyclopedias (britannica), literature tools (citavi, zotero), reading and writing support for other document formats (e.g. Open Office) and more[9].

The Microsoft Store for Office apps listed 192 additional components at the time. However, this may be only the tip of the iceberg. Many add-ins are developed and distributed independently by their creators. The community is highly decentralized, with a common platform that however does not allow for interaction (Kahler 2001a).

3.3.4. Sketchup 3D modeling tool

Sketchup is a 3D modeling software, for *"architects, designers, builders, makers and engineers"*[10]. Compared with other CAD tools, it is considered very easy to use.

Sketchup can be tailored by modifying the toolbar to add, remove or relocate icons. Additionally, further tool palettes can be shown which add specific supplementary functionalities. Moreover, Sketchup can be expanded by so-called extensions. These small components add supplementary functionalities as support for different file formats (e.g. STL or OBJ), rendering images, 3D printing objects, etc.

For several years, Sketchup was at the center of a very active community of extension users and developers, loosely organized within a commonly used forum. Currently, the so-called Sketchup extension warehouse[11] is the starting point for users to pick and install their extensions (see Figure 5). The extension warehouse currently lists 267 components.

9. http://office.microsoft.com/en-us/store/?CTT=97 (last accessed 2014/05/16)

10. http://www.sketchup.com/about/sketchup-story (last accessed 2014/05/16)

11. http://extensions.sketchup.com/en/search/site (last accessed 2014/05/17)

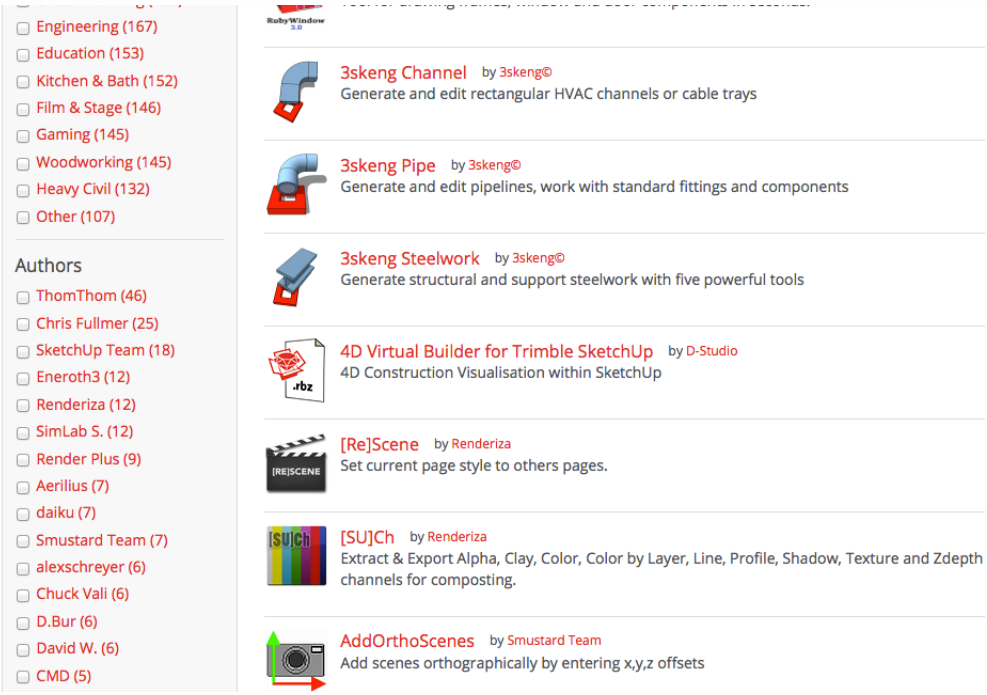

Figure 5: The Sketchup extension warehouse, showing categorization mechanisms for searching (left), as well as several extensions (right). (http://extensions.sketchup.com/en/search/site accessed 2014/05/17)

3.3.5. Eclipse - Integrated Development Environment

Eclipse is an integrated software development environment (IDE), a software engineering toolbox, used by programmers, software architects and testers. It started out as a development environment for the Java programming language but over the years it was expanded to cope with many major programming languages and software engineering technologies. Eclipse was created by IBM in 2005. Later it was made open source and is currently being developed by the Eclipse foundation (O'Mahony, Diaz & Mamas 2005). That said, the foundation is very small and only engages in coordination work. All implementation work is carried out by large corporations to small businesses and hobbyists, who contribute their work to Eclipse.

The Eclipse user interface can be tailored in several ways, but most importantly, so-called plug-ins can be installed which result in new functionalities. Examples are support for additional software engineering tools (e.g. bug trackers, code repositories or additional programming languages). Furthermore, Eclipse and the additional plug-ins can be configured in several ways, to fit the project under development (Gamma & Beck 2004).

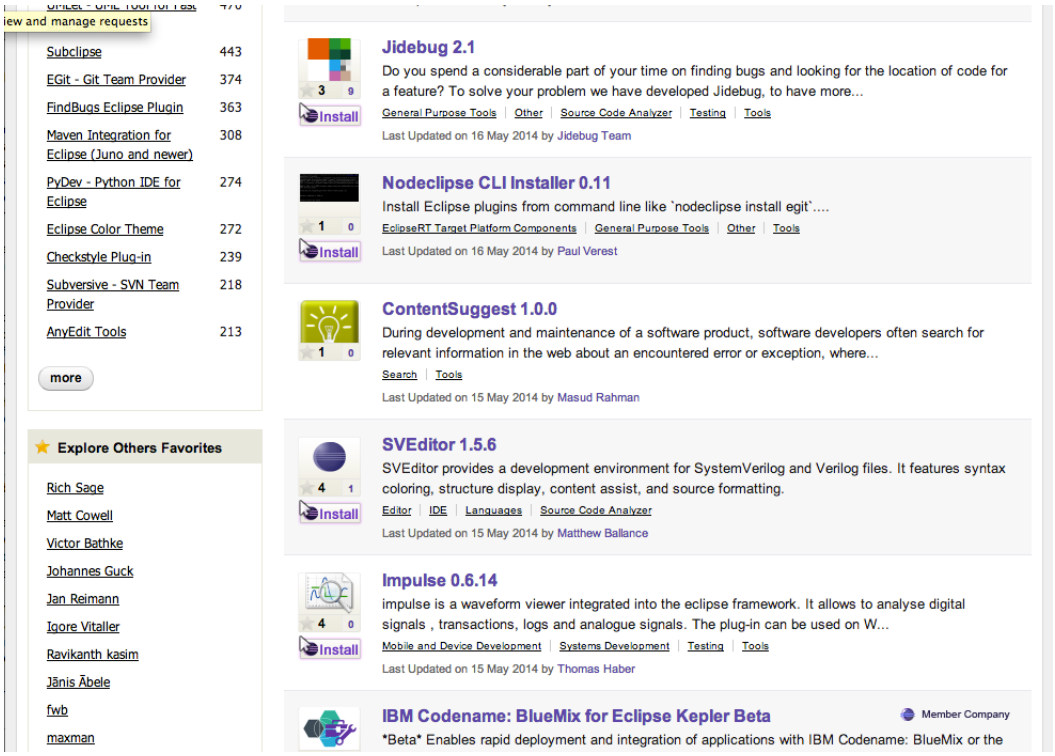

Figure 6: Eclipse Marketplace, showing several tools for installation into Eclipse (right), as well as categories to search for certain components. (http://marketplace.eclipse.org/ accessed 2014/05/16)

The Eclipse community is quite large and heterogeneous. Some means for centralization exist, like the Eclipse forum for experience exchange and further development of Eclipse and the marketplace for components, which currently lists 1,751 components (see Figure 6). However, many additional components are not well represented there. The community is very powerful, but also very complex (Gamma & Beck 2004).

3.3.6. World of Warcraft

World of Warcraft (often abbreviated as WoW) is a massive multiplayer online role playing game that was very successful in recent years with about 11.5 million users. World of Warcraft is developed and licensed by Blizzard entertainment, one of the largest game design companies in the world (Nardi 2010).

The game can be extended with various add-ons, which deliver new features. As Blizzard wants to balance gameplay and prevent unfair advantage over other players, these features and therefore the power of add-ons is limited to changing just the user interface and the display of values that are relevant for the gameplay. Some of these add-ons can be configured afterwards (Nardi 2010).

World of Warcraft add-ons (which currently number 6,496) are at the center of a 3rd party community, driven by Curse gaming (see Figure 7)[12]. They offer features similar to Firefox or Photoshop, as they present popular add-ons and let users rate add-ons, etc. However, there seems to be an important personal exchange between collaborating players on which add-ons to use.

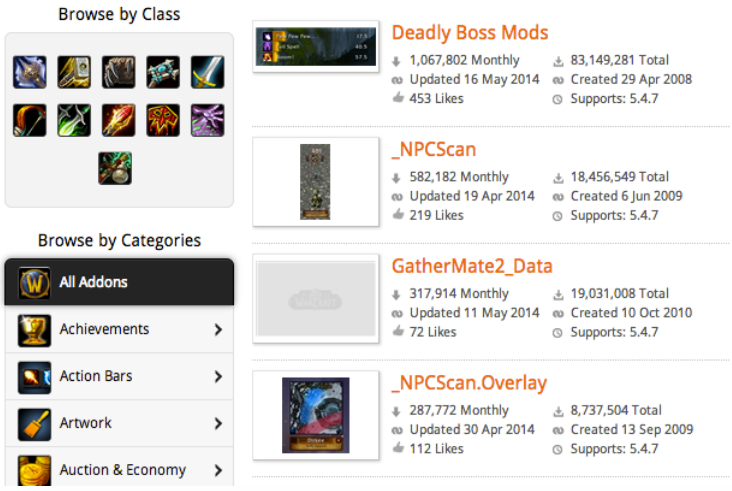

Figure 7: Partial list of World of Warcraft add-ons at Curse.com. (http://www.curse.com/addons/wow accessed 2014/05/17)

3.3.7. Discussion

All of the introduced software systems are an interesting base to investigate the presented research questions. However, the Eclipse IDE and World of Warcraft seem most promising. Together with Photoshop, they are arguably the largest in terms of existing components. Unlike Photoshop, their components are very easy to access. Photoshop plug-ins are often costly, for example; therefore a practical investigation would have been much more difficult.

At the same time, we can assume a certain level of collaboration and potential needs for an exchange of experiences between users. Programmers using Eclipse very often work in work groups or project teams. World of Warcraft players organize themselves in clans – groups of players that work together in order to achieve better results. The same cannot be said for users of other systems. Firefox and Office usage is very often not as collaborative.

In the end, the Eclipse Integrated Development Environment was chosen from this group as field for research. We recognized some phenomena as problems with Eclipse

12. http://www.curse.com/addons/wow (last accessed 2014/04/10)

usage and maintenance during informal interviews with Eclipse users that attracted our interest. Furthermore, the pragmatics of field research, lead to a stronger focus on the Eclipse IDE. However, the following additional considerations show that the decision to choose Eclipse as field for research was an excellent decision:

The *technical possibilities to tailor and extend Eclipse* are much more extensive than the tailoring possibilities in Sketchup and World of Warcraft. The plug-in mechanism (Gamma & Beck 2004) established Eclipse as one of the most flexible software systems today. Figure 11 (see page 122) illustrates this very clearly by comparing it to a more typical component architecture (included in this comparison).

Eclipse's *user or target group* consists of software developers, architects and testers. This target group is of special interest for appropriation research. It can be argued that they embody many of the skills that other users may have and need in order to cope with flexible systems. They are generally considered to be technically skilled and are regarded as being able to solve technical problems very well. Incompatibilities, updates, or more typically making software work as part of appropriation can be counted amongst their their skills.

They rank as being very open-minded regarding the adoption of new technology (Strübing 1992), which is an important factor from a research pragmatics point of view. Appropriation is enmeshed in other practices that aren't carried out every day. As such it lacks *observability* in the wild. Due to this, every aspect that increases the chance of triggering such activities also increases the chance to gather data for this study.

The general field of application (software development) is a rather complex and usually group based activity. This potentially provides good general conditions for observing the *collaborative effects* of appropriation.

Eclipse – considered to be an open source development project – is conceived as an impeccable example of open source software. This has come about not only due to the relative stability of the software but also due to the emergence of a huge *community* of users which has been continuously growing since 2001 (O'Mahony, Diaz & Mamas 2005).

One result of this community is the existence of more than 1,751 tools that are largely free of charge and can extend the delivered functionality of Eclipse (see section 3.3.5). Despite the existence of a common marketplace, their distribution is very heterogeneous. Every actor contributing a plug-in can choose to deliver it as he likes. Forms of distribution are:

- Packaging and providing the plug-ins (or rather the container format of features) as an archive for download (zip file). Such components can be copied directly into the Eclipse installation or installed using the update manager.
- Providing an update-site that supports the Eclipse update manager. This option is superior, as the update-site can be automatically checked for updates later.
- Providing an update-site and listing the plug-in in the Eclipse marketplace.

This creates a complex situation for users that reflects the *missing integration responsibility* within software ecosystems very well.

Concerning *access to the research field*, Eclipse is again a good choice. At the time of the study, Eclipse was perhaps the most important development environment for the Java programming language, as well as being a well-known environment for other languages (Vaughan-Nichols 2003; Geer 2005). The large market share of Eclipse allowed us to easily access Eclipse users for the field study. It was a crucial aspect as this study was laid out to approach work groups in organizations which were as diverse as possible.

The Eclipse IDE is open source at its core. Focusing on Eclipse therefore allows for design prototypes that would not be possible when relying on other software systems. This could facilitate the design of appropriation support that could be integrated and tested directly as part of Eclipse.

3.4. Research method

The overall case study of this work largely followed a praxeological approach (Reckwitz 2002; Wulf et al. 2011). It focused on the practice level of users as a starting point for the investigation of the social and socio-technical phenomena around Eclipse appropriation. Underlying this idea is an existing definition that clarifies the focus on practices further:

"Practices are understood as the smallest unit in the analysis of social phenomena. A practice is understood to be a mainly routinized pattern of human action which is not only encompassed by mental and physical forms of activity but that is also greatly imprinted by objects, especially by tools, media, and their usage. A practice is grounded in background knowledge that is both not entirely explicit and containing emotional as well as motivational elements. Practices, therefore, represent collective patterns of interaction that are reproduced in specific contexts. While the collective patterns of interaction are routinized, the concrete action is situated context-specifically and may deviate from them." (Wulf et al. 2011)

This understanding of practice and idea to theorize about practices, allows appropriation to be understood as originating from its visible, observable part: human practice. Originating at this point allows additional questions regarding the involved human agents and artifacts to be asked. In such ways personal and collaborative underlying motivations and values as well as situational contexts within groups or organizations can be pursued, interpreted and connected. Or as summarized by Suchman: *"To understand technologies ethnographically, it is required that we locate artifacts within the sites and the relations of their everyday use."* (Suchman et al. 1999)

Concerning the design of supporting technology (see part III), a grounded design approach was followed (Stevens 2009; Ramírez Zúñiga 2012). Grounded design as carried out during this work can be classified as follows: During the (long running) design process, very few presumptions about the field were used to define a prototypical design. Instead, ideas for design had been grounded in the results of the empirical field research. Furthermore, some of the concepts were open for discussion and presented to users in order to gather feedback and include potential users into the design process (Draxler, Sander & Stevens 2010). The design was influenced and inspired by, but was not clearly a result of, the field study itself. The design process was intricately entangled with the field research, as design ideas were grounded in (or discussed against the) results of the field study. The combination of the empirical field study grounded design led to a design concept and later implementation which had not been created under previously defined hypotheses, rather the main conceptual aspects of the design stemming from the empirical field work. The actual activity of designing in the grounded design approach is similar to the activity of analyzing as part of the grounded theory approach (Glaser & Strauss 1967) (see also section 3.4.2). Both approaches demand further investigation, until theoretical saturation is reached. Theoretical saturation in grounded theory is the point where the additional analysis of new material does not raise new results (Strauss & Corbin 1996). In grounded design it can be considered the point where additional field research does not contribute to further describe the design space, which is to be filled creatively through designing.

To some extent, the presented work can be also seen as a mixed method approach (Kelle 2001). The first investigation made use of a questionnaire that was evaluated mainly in order to determine the rate of participants working in a collaborative environment, sharing tailoring artifacts, etc. The following investigation adopted the micro-level approach, as previously introduced.

3.4.1. Gathering data

The first part of this study was carried out as an exploratory interview study with Eclipse IDE users. While the results have not been published, they sharpened the view

towards this phenomenon: Eclipse users were struggling with the maintenance, configuration and appropriation of their main working tool.

A first investigation into the field of Eclipse usage and appropriation was prepared as a study based on an online-questionnaire in order to understand if modification and exchange was common among Eclipse IDE users. The results of this work are presented in chapter 4. The question and answer categories of this questionnaire were prepared, based on the first results of the exploratory study. The questionnaire was published in several well-known places for Eclipse IDE users (e.g. forums consulted by web developers). The questions were evaluated, using statistical analysis (e.g. count of installed plug-ins, top 10 installed plug-ins, most used versions of Eclipse IDE, etc.). This was especially interesting, as we did not only ask certain questions, but requested the participants to attach a footprint of their Eclipse IDE. All participants found this easy to achieve as Eclipse provides a function to automatically create such footprints as a text file, including version numbers and installed plug-ins.

However, the most useful way to investigate the proposed research questions was by conducting an empirical field study. Over time, we gained access to seven organizations that at the time were using the Eclipse IDE. A detailed introduction is given in section 3.5, an overview of the complete material is shown in Table 6 on page 119. We were allowed to carry out observations, interviews and workplace investigations combined with small ethnographical interviews (Randall, Harper & Rouncefield 2005). Furthermore, we gathered artifacts. All interviews were recorded and transcribed for later analysis. Screenshots and Eclipse footprints were taken, during workplace investigations. In the course of these observations, we furthermore created field notes.

As part of a comparison of Eclipse appropriation with the appropriation of another flexible software system, we chose World of Warcraft as a second field for research (see section 3.3.6). We applied the same data gathering method (and analysis method) to a group of World of Warcraft players, in order to compare the results (see chapter 7).

3.4.2. Data analysis

The detailed analysis of the material gathered during field visits was carried out by following the strategy of theoretical sampling as suggested by the grounded theory approach (Strauss & Corbin 1997). Therefore, the data was coded (in grounded theory terms: breaking the surface of the material (Strauss & Corbin 1997)). Over time, through continuous comparison of codes and by carving out the characteristics of codes, larger categories were formed which span multiple codes and present a less detailed but easier to comprehend unit. In order to achieve this, the grounded theory ap-

proach instructs that leading questions should be asked. These can be classified into several categories to systematically explore the data:

"(A) conditions -> (B) phenomenon -> (C) context -> (D) intervening conditions -> (E) strategies of action -> (F) consequences" (Strauss & Corbin 1996, p. 78).

Data from each organization was first investigated separately, in order to search for differences between the participating organizations. During this phase, the Eclipse IDE footprints were also analyzed. A small tool was prepared (see Figure 8) to pick interesting data from the footprints and export it as files, processable by spread sheet software in order to gain an overview of the data and create graphs that represent certain aspects of the data. This data was used to validate and add details to statements, given during the interviews.

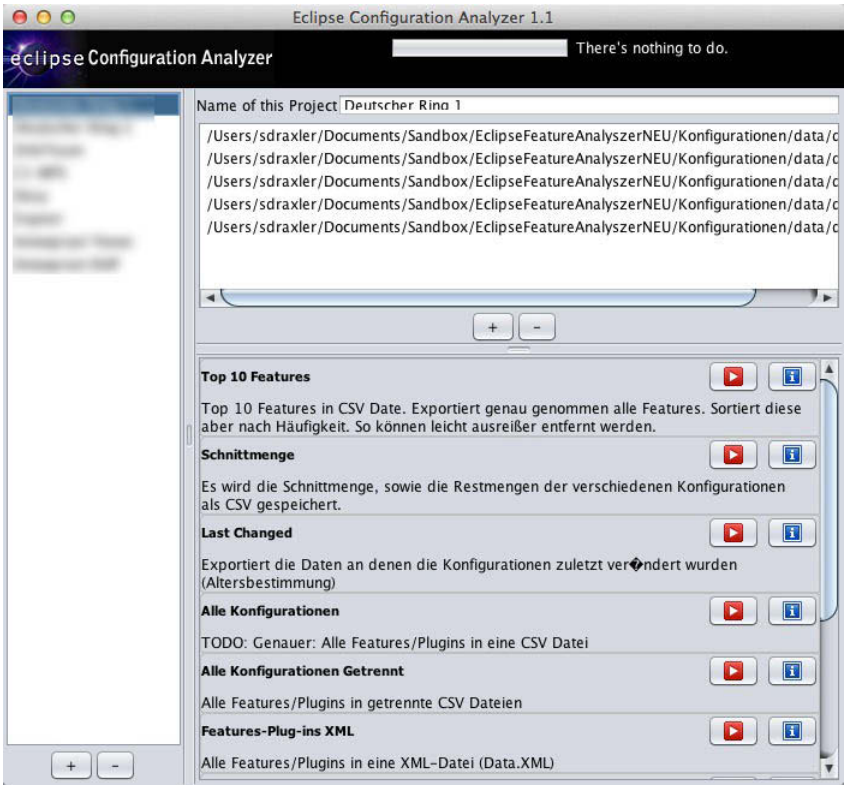

Figure 8: Eclipse Footprint analysis tool.

The organizations were carefully selected to ensure they differed from each other. This was part of the effort to achieve a theoretical saturation (Strauss & Corbin 1997) during the analysis. The grounded theory approach suggests selecting cases which differ from each other in order to gain a range of insights during the analysis, up to the point where new different cases do not provide any new results (theoretical saturation). Despite careful selection, the results proved to be very similar, thus rendering a side by side comparison unsuitable. Instead, a more comprehensive analysis compris-

ing all data was carried out. Furthermore, theoretical saturation was reached after investigating seven organizations.

As part of this analysis, we also investigated the socio-technical structure of the Eclipse ecosystem since this determines users' opportunities to shape their personal workspaces. The work of other researchers was mainly included subsequent to the original analysis in order to describe certain uncovered phenomena using naming conventions known in the scientific CSCW and HCI community.

3.4.3. Ethnographically informed design

Parallel to field visits and interviews, different design interventions were carried out. This included participatory design (Schuler & Namioka 1993) workshops with Eclipse users, sketching (Buxton 2007) workshops with designers, reflection workshops to ground ideas within the results of the field study and evaluation (Dahlbäck, Jönsson & Ahrenberg 1993) workshops with Eclipse users. Overall, these steps were carried out at various times, not strictly in this chronological order during later iterations.

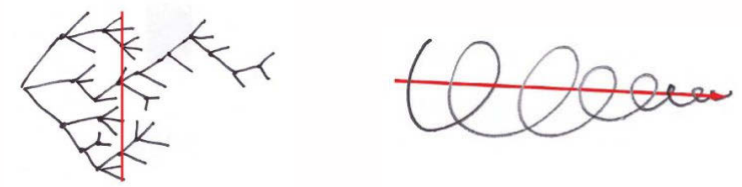

Figure 9: Left: prototyping-approach; right: sketching-approach (Buxton 2007)

The early process was organized as a sketching approach (Buxton 2007) (see Figure 9 left) in order to envision ideas. Those early concept ideas followed a recommender system approach for customizations as well as a social network for customizations and experience exchange. However, the ideas were quickly discarded in favor of an awareness approach (Heath, Luff & Cambridge 1992) that was better grounded in the empirical results and showed potential during a wizard of oz evaluation (Buxton 2007; Dahlbäck, Jönsson & Ahrenberg 1993). From this point onwards, the process changed to a prototyping approach (Buxton 2007) (see Figure 9, right), focusing more on making this design concept usable for support, rather than arguing for different concepts.

The next phase saw an improvement in interaction with the demonstrator. A participatory design workshop with regular Eclipse users helped to shape the provided information and interaction design in that respect.

Additionally, the prototype was presented to seven Eclipse users from two of the organizations participating in the study. First, early versions of screenshots were used but as the prototype improved, the complete software was shown in order to evaluate

the user interface. Based on this, the prototype was stabilized, bugs were fixed and the user interface received a cleanup. It was finally tested (Wulf et al. 2011) and discussed with eight users at a research institute, who integrated the plug-in into their daily working environments (see chapter 9 for more details).

3.5. Overview of Eclipse-related field visits

The main part of this work is based on a corpus of empirical field studies, carried out in seven different organizations. At the time, all organizations used Eclipse for certain software development efforts and agreed to participate in a study to investigate their Eclipse IDE usage. To contextualize the results presented in parts II and III, the participating organizations are introduced in this section. Please note that the data for these descriptions was gathered in the course of the field study. The names of the organizations, products and services as well as the names of the participants have been replaced with pseudonyms. Please also note, that a much more detailed overview is presented in Appendix I.

- **Alpha:** Is a small German software producer (about 10 people). Their main asset is a groupware system that is licensed to customers. The main tasks can be described as further developing the groupware system for customers as well as maintenance and bug fixes. There was a divide between the senior staff and the junior workers. Seniors preferred command line tools, while the junior developers preferred the Eclipse IDE. Interviews and workplace visits with all Eclipse users were carried out.
- **Beta:** Is a small German web developer (8 employees + freelancers). Their work consists mostly of the development of small web-based applications that incorporate Open Source Software. At the time, they used the Eclipse IDE in most projects. Interviews with the CTO and one junior developer have been carried out. to represent a senior and a junior developer.
- **Gamma:** Represents the software development department at an insurance company, responsible for in-house development. The whole company employs 1,200 people, the development department consists of 90 people. They mostly develop Java-based web-applications. Interviews with three members of one development group were carried out. We chose this group, as the project leader was actually an employee of Delta and we hoped to get some interesting insights, because of that. The group had to cope with certain problems, because of a very restrictive Firewall, that prevented updates and download of plug-ins.

- **Delta:** Is a German software development and technology consultancy company, about 50 people strong. Most employees are specialists in using Eclipse and in developing new software that is based on Eclipse technology. We interviewed two people, typical for the company at their work places.
- **Epsilon:** Represents the maintenance group at a German 1,700 people research facility. The maintenance consists of 20 people for software and hardware development. They are responsible for the development and maintenance of the cooling facility of a particle accelerator. We carried out three interviews with typical staff members and visited their work places.
- **Zeta:** Is a medium-sized (300 employees) German enterprise, known for small and medium sized web-applications. At the time, they used the Eclipse IDE for nearly all projects. Their trouble with the maintenance of Eclipse, lead to making one member of the company responsible for Eclipse IDE management. We interviewed four experienced Eclipse users, that were part of one development team.
- **Theta:** Is a small German software development company, employing about 30 people. Theta is an expert in Eclipse technology (similar to Delta), as their two main products are based on Eclipse technology and developed using the Eclipse IDE. We interviewed the project leader of ThetaProduct as well as two of his developers at their work places.

While the beginning of chapter 4 is based on a questionnaire, the main part is based on research carried out at Alpha. Chapter 5 is based on data from several, however not all organizations (see Table 5). Chapter 6 is based on all of the introduced organizations Alpha to Theta. Chapter 7 is based on a subset of these organizations (Alpha to Epsilon). The data used for chapter 7 was supplemented by several interviews with, as well as remote participant observations of a group of World of Warcraft players. This was carried out to gather comparable data, which is described in chapter 7. Chapter 8 was also based on the Data of Alpha to Epsilon, while chapter 9 again makes use of the whole dataset.

3.6. Mapping of sections and research questions

Part II of this book will center on discussing *Q1* and *Q2,* outlined above. The four chapters (4-7) have already been published and resemble the accepted versions of the related journal or conference papers (see Related Publications on page V).

- Chapter 4 provides an overview over some aspects of appropriation of the Eclipse IDE, as well as a broader study showing that the observed phenomena are not only specific for the selected research sites. It outlines the dilemma between up-

dtating Eclipse and stabilizing Eclipse. Furthermore, it presents a detailed analysis of Alpha.

- Chapter 5 focuses on a group view of the observed practices. It centers on the definition of types of collaboration within organizations or work groups during appropriation efforts.
- Chapter 6 presents a very detailed analysis of observed appropriation practices and contextualizing situations. It connects both aspects and creates a rich understanding of the appropriation of Eclipse, grounding the presented categories in the users' work and organizational specifics in order to relate the results to other fields.
- Chapter 7 focuses mainly on Q2, as this presents a different field study which investigated the field of World of Warcraft players in order to compare appropriation between them and previous results from the Eclipse IDE case study.

Part III of this book will introduce results to discuss *Q3* outlined above. Both chapters have already been published and resemble the accepted versions of the related conference papers (see Related Publications on page V).

- Chapter 8 provides an early overview of the design space, concerning the support of Eclipse appropriation.
- Chapter 9 presents in detail several design-principles that have been evaluated and implemented for prototypical demonstration. This exemplifies and demonstrates a new category of appropriation support: relying on the group instead of communities and awareness as a main channel.

Part IV summarizes the results of this work, discussing them against the backdrop of the presented related literature and points out findings that go beyond the state of the art. Furthermore, open questions that could not be answered or have risen during this work are discussed.

II.

Understanding Eclipse Appropriation

4. The Collaborative Appropriation of an Open Software Ecosystem[13]

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.

13. This chapter has been published as: Draxler, Sebastian and Stevens, Gunnar, 2011. Supporting the Collaborative Appropriation of an Open Software Ecosystem. Journal of Computer Supported Cooperative Work (CSCW), Volume 20, Issue 4-5, pp. 403–448, October 2011. With kind permission from Springer Science and Business Media. http://link.springer.com/article/10.1007%2Fs10606-011-9148-9

4.1. 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 1990a; MacLean et al. 1990; Mørch 1997; Kahler 2001b; 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 (1992) 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.

Messerschmidt 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 (Boucharas, Jansen & Brinkkemper 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 & 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 4.2 gives an introduction into the related research on workplace design and appropriation work. Section 4.3 outlines the mixed method research approach we applied, discussing the relation between the ecosystem analysis, the survey and ethnographically oriented study in detail. Section 4.4 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 4.5 presents the findings of the survey, contributing to the second research question. Section 4.6 presents the findings of the in depth case study, answering the third question. Section 4.7 interprets the empirical findings in terms of technological and organizational opportunities to support the appropriation of software ecosystems. In section 4.8 we give a conclusion and we discuss the transferability of our findings in more detail.

4.2. Workplace design as “artful integration”

In this section we want to draw a rich picture of designing workplaces 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.

4.2.1. 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 & Carroll 2000; Venkatesh & 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 & 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, Gilbert & Luff 1992).

With regard to the design of computerized workplaces we found such permeation in the common standards for IT management like ITIL and CobiT (Bon 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).

4.2.2. 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 & Kyng 1992; Wulf 1994; Muller, Haslwanter & Dayton 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, Quaisser & Klann 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 & 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 & Kyng 1992; 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, Lai & Fry 1995), Prospero (Dourish 1996), DODE (Fischer 1994) or FreEvolve (Wulf, Pipek & Won 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.

4.2.3. Patterns of sharing customizable working environments

A large body of literature has considered tailoring to be an individual effort (cf. Friedrich, & Rödiger 1991) and research on tailoring support therefore mainly focused on personalization (i.a. Kobsa & Wahlster 1989; Oppermann 1994). This focus shifted in reaction to empirical studies that uncovered the collaborative dimension of tailoring (i.a. Mackay 1990a; MacLean et al. 1990; Gantt & Nardi 1992; Wulf 1999b; Kahler 2001b). 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 (1990a). 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 Theory (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 (2001b) 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.

4.2.4. From tailoring to appropriation research

In the last ten 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 & Haddon 1996; Dourish 2003; Pipek 2005; Balka & 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. Markus 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 (Ruel 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 (Bertell 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 & Wagner 2006). Technology should therefore provide an

outer-tailorability (Pipek & 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 & 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.

4.2.5. Managing the coevolution of artifacts within Software Ecosystems

The term software ecosystem refers to a new software production paradigm in Software Engineering (Messerschmitt & 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 & Bosch-Sijtsema 2010a); methods to specify the basic architectural structure that (implicitly) defines what is fixed and what is adaptable or extendable (Dittrich, Lindeberg & Lundberg 2006); mechanisms to coordinate globally distributed software development (Crowston et al. 2008; Bosch & Bosch-Sijtsema 2010b); dealing with the collaboration and competition at the same time (Henkel 2004; Jansen, Finkelstein & Brinkkemper 2009); and building and managing a software ecosystem around a product including a community of external developers, domain experts and users (Bosch & 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 soft-

ware will increasingly be mixed and interlaced with the tailoring (Dittrich, Lindeberg & Lundberg 2006; Eriksson & Dittrich 2007; Dittrich, Vaucouleur & Giff 2009).

Part of the design problem is the difficulty to anticipate changes for which to provide (Dittrich, Lindeberg & Lundberg 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, Lindeberg & Lundberg 2006). Beyond an organizational tailoring culture, therefore, a cross-boundary culture of participation is needed (Eriksson & 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 & Mørch 2009).

4.2.6. 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 & Ruhleder 1994; Karasti, Baker & Halkola 2006; Pipek & 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 (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 & 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 & 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, Baker & Halkola 2006; Karasti, Baker & Millerand 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 two 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 el 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.

4.2.7. 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.

4.3. Methodology

Since the 80s there is a growing market of tools In the software industry supporting the various tasks like compiling, debugging, code control, etc. (Chikofsky 1989; Fisher 1991). But using the diverse tools together was clumsy and error-prone. Permeated by tayloristic thinking, in the 90s 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 (McClure 1989). However, despite great efforts in design and research the paradigm failed to realize the promises (Sommerville 2007) and did not reach acceptance of the practitioners (Elshazly & Gover 1993; Iivari 1996; Chervany & Lending 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 1990a) 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 & Allert 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 4.6.

4.4. 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.

4.4.1. 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, Diaz & Mamas 2005), where a distributed development process has to be coordinated (Grinter, Herbsleb & Perry 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, Herbsleb & Perry 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, Diaz & Mamas 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 poweredby-pulse.com are specialized in maintaining repositories of 3rd 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.

4.4.2. "Everything is a plug-in": 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 imple-

ments 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 (Gamma & Beck 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, Diaz & Mamas 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"* (Gamma & Beck 2004).

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 extend-

ing 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.

4.4.3. 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.

4.4.4. 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 (Gamma & Beck 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 & 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.

4.5. 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 & Wagner 2006). By its nature this artful integration is almost invisible and difficult to quantity (Star & Ruhleder 1994; Dörner, Heß & Pipek 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.

4.5.1. 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.[14] 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 3). The standard deviation of features $\sigma f=36.8$ 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.[15]

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,

14. A feature in Eclipse defines a set of plug-ins and sub features which must be installed when the feature is installed.

15. We calculated the distance of two installations with the set of features Ci and Cj as follows: $ufeature(Ci,Cj) = (|Ci\backslash Cj|+| Cj\backslash Ci|)/(|Ci|+|Cj|)$. Based on this calculated the average distance: $\bar{u}feature(C1,\ldots, Cn) = 1/n*(n-1)*\Sigma 0 \leq i<j \leq n$ ufeature (Ci,Cj). 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.

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.

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.

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

Table 3: Amount of plug-ins found in Eclipse installations (with n=76 Eclipse installations).

4.5.2. 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.

4.5.3. 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).

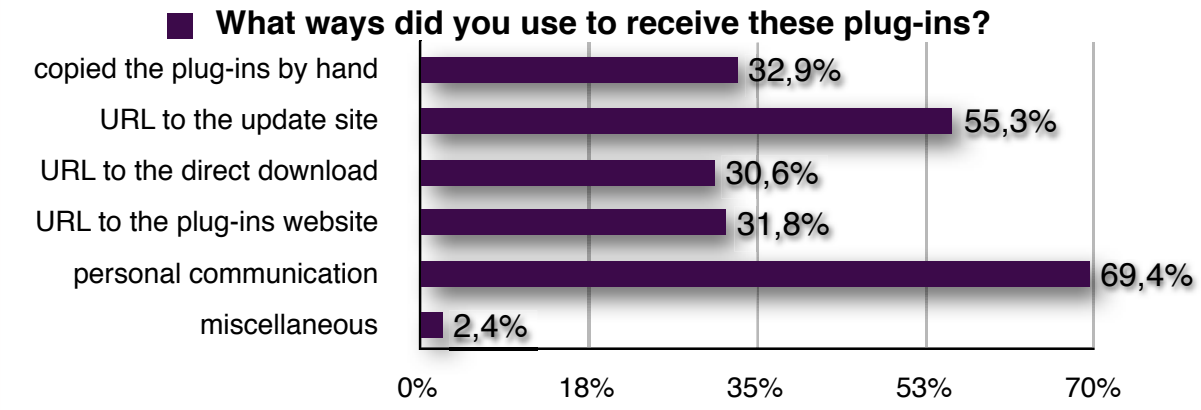

Figure 10: Channels used to receive plug-ins and plug-in information. The other way round *"Did you ever share plug-ins with colleagues?"* and *"Which way did you use to share these plug-ins?"* provide nearly identical pictures.

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 10 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 im-

portant 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.

4.5.4. 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 & Ruhleder 1996; Balka & Wagner 2006). Moreover, the survey demonstrated to what extend adapting Eclipse became part of ordinary work activities and ordinary work situations.

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 1990a). 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.

4.6. 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 4 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.

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.

Table 4: List of situations that contextualized appropriation work.

4.6.1. Organizational context

Alpha is a small software company, which is quite typical for German software industry. The company was founded twenty 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 customer 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.

4.6.1.1. 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 (Bon 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.

4.6.1.2. 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 hours 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 1998) 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)

4.6.2. 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.

4.6.2.1. 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.

4.6.2.2. 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 & 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.

4.6.2.3. 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)

4.6.2.4. 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)

4.6.2.5. 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)

4.6.2.6. 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 & 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 & Bowker 2006; Karasti, Baker & Millerand 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 4.4). 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 tech-

nology" 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 4.5) and the previous research on collaborative tailoring (Mackay 1990a; 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 1998). 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, Baker & Halkola 2006; Karasti, Baker & Millerand 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. MurphyHill 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"* (Beck 1999). 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.

4.7. 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 & Piller 2002). 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.

4.7.1. 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.

4.7.1.1. 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 4.4). 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 4.7.2 within supporting the tool awareness among colleagues.

4.7.1.2. 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 instal-

lation, 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 4.6.2.4). 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 & Golombek 2001).

4.7.1.3. 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 2001b; Wulf & 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.

4.7.2. 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 1990a). 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 (1990a), 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 4.6.2 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 (2001b) 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 & Wiedenhöfer 2006).

To support need driven appropriation, search based user interfaces and recommender techniques seem promising. Concepts like the Expert Finder (Reichling, Veith & Wulf 2007), 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) (see chapters 8 and 9). The intention of Peerclipse is to re-

spect 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.

4.7.3. 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 (2001b) 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 4.3 and 4.4). 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.

4.8. 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 1990a) 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 & 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, Baker & Millerand 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 (Hartmann et al. 1994; Kahler 2001b; 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 & 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, Pipek & Wulf 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 4.4) 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 4.7 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.

5. Team Practices of Appropriation in Software Ecosystems[16]

Since the 1990s, the forms of production, distribution, configuration and appropriation of software have changed fundamentally. Nowadays, software is often embedded in *software ecosystems*, i.e. in complex interrelations between different stakeholders who are connected by a shared technological platform. In our paper, we investigate how small software teams deal with the challenges of appropriating and configuring software in the Eclipse ecosystem for their daily work. We empirically identify three different approaches for dealing with appropriation in software ecosystems which are represented by the "ideal types" *lone warrior*, *centralized organization*, and *collegial collaboration*. Based on a discussion of these strategies and the underlying appropriation practices we found in the field, we suggest theoretical and practical implications for supporting appropriation in software ecosystems.

16. This chapter has been published as: Draxler, Sebastian; Jung, Adrian; Boden, Alexander; Stevens, Gunnar; 2011. Proceeding of the 4th international workshop on Cooperative and human aspects of software engineering, Workplace warriors: Identifying Team Practices of Appropriation in Software Ecosystems. ACM, New York pp. 57-60. © 2011 ACM, Inc. http://doi.acm.org/10.1145/1984642.1984656

5.1. Introduction

For a long time, research on software usage has been interested in aspects of customizing and tailoring single applications to the needs of end users (Gantt & Nardi 1992; MacLean et al. 1990; Wulf 1999b). This was well suited in times when the software market was limited and applications had a clear border. However, several trends have changed the ways of software production, distribution and configuration considerably over the last two decades: 1.) the establishment of the Internet as the dominating infrastructure for disseminating and marketing digital goods; 2.) the spread of new business and development models in the Open Source domain, encouraging users to share software with others; 3.) the establishment of software ecosystems which consist of different manufacturers and hobbyists, creating small-scale components that can be individually assembled by end users. These developments do not only affect the ways how software is developed and maintained, but also how it is appropriated by the users (who are increasingly understood as prosumers instead of mere consumers of software). As the social and collaborative aspects of appropriation in software ecosystems are not well understood, we have conducted a study which investigated how small teams of software developers deal with the necessities of appropriating the software that constitutes their work tools—in this case the Eclipse IDE and its surrounding software ecosystem (Eclipse Foundation 2010).

The paper is organized as follows: after a discussion of the related work and our methodology, we provide an overview on three different approaches of dealing with appropriation as well as the underlying practices and implications that constitute these strategies. Based on our findings, we then identify implications for the theoretic understanding of appropriation as well as for designing supportive tools for these practices.

5.2. A Brief History of Appropriation

In the 80s and 90s, the research on the social and socio-technical aspects of software use and development coined the terms *customization*, *adaptability* and *tailorability*. Driver of these discussions was the fact that the existing, monolithic off-the-shelf products did not accord well with the complex, heterogeneous needs of a dynamic market. While tailoring was often considered to be an individual effort, later research highlighted the collaborative aspects of these practices. A major contribution to this shift was the work of Mackay (1990a), who conducted empirical studies within organizations to examine patterns of sharing self-created email filter rules as well as configurations of Unix desktop systems. In a similar vein, Gantt and Nardi (1992) investigat-

ed into the tailoring practices of CAD users. Under their lens, collaborative efforts became visible. While tailoring has been mainly discussed from a technical perspective, recent research has broadened the understanding of the corresponding practices by analyzing the ways how users fit the technology at hand into both the pre-existing culture and into the local patterns of use and life rhythms (Stevens, Pipek & Wulf 2009). Conceptually labeled as *appropriation*, this line of research highlights the link between the creative re-configuration and the re-interpretation of given features as two dialectically connected forms of end user development (Dourish 2003; Stevens 2009), implying that technical concepts as tailorability (in terms of re-configuration of software) should be supplemented by social-technical means for supporting appropriation (Dourish 2003; Pipek 2005; Stevens 2009). Generally speaking, while tailoring focuses on the technical customizations of software artifacts, appropriation also considers the social-technical process of interpreting software in daily work practice. While research in this domain has predominantly focused on the appropriation of single applications so far, we want to understand how software users design their workplaces by assembling heterogeneous resources from global software ecosystems.

5.3. Research Methodology

From 2006 to 2009, we participated in a publicly funded research project to develop tailorable applications for different domains (like monitoring complex technical plants, eLearning based on business simulation and using basic groupware technology). The aim was to study the potential of the Eclipse platform (Gamma & Beck 2003) as a core technology for realizing the concept of component based tailorability (Mørch et al. 2004). Eclipse was chosen as a platform as it was considered being a leading edge technology in respect to a consequent component architecture ("everything is a plug-in" (Gamma & Beck 2003, p. 83)) and a vital ecosystem (with thousands of free or commercial extensions). We expected these features to empower users to personalize their applications by managing individual plugin portfolios, selected form the Eclipse software ecosystem.

Our research was organized following the strategy of theoretical sampling as suggested by Strauss' Grounded Theory (Glaser & Strauss 1967). We started with an open-ended, qualitative study on Eclipse plug-in adoption practices, using semi-structured interviews and on-site observations. Specifically, we cooperated with five German companies with 10 to over 2000 employees that perform software development (see Table 5 for details). In each company, we conducted at least three semi-structured interviews of at least one hour (altogether, we conducted 18 interviews between August 2007 and

December 2008). Additionally, we visited two SMEs over a period of 3-5 days for on-site observation.

All interviews were audio recorded and transcribed. The participant observations were documented by means of field notes taken during the research. In addition, we also analyzed the socio-technical structure of the Eclipse ecosystem, which determines the possibilities of users for designing their personal workspace. Related work done by other researchers was investigated for the purpose of sensitizing our analytic lens. However, we tried to prevent to subsume our observations under pre-defined concepts taken from literature. Instead we focused on categorizing our data by carefully analyzing the documented practices in minute detail. In a further step, we supplemented our qualitative analysis by an online survey where we used to assess the spreading of the observed practices (Stevens & Draxler 2010).

	Employees	**Domain**	**Collected Data**
Alpha	10	Software Development	3 Interviews
Beta	6 + freelancers	Software Development	3 interviews 3 day observation
Gamma	1,200 (90 developers)	Insurance company + inhouse software dev.	3 interviews
Epsilon	1,700 (30 maintenance dept.)	Research facility, Inhouse development for machine maintenance	4 Interviews
Zeta	250	Development of Multimedia and Web applications	5 day observation 5 interviews

Table 5: Research Partners.

5.4. A Classification Scheme of Team Practices

Our research provided us with rich insights into the manifold practices of how practitioners of small software teams appropriate software in the context of the Eclipse ecosystem. In general, we observed that practitioners were in a constant struggle of innovating their working environment and keeping it up-to-date in order to perform their work. At the same time they had to prevent possible breakdowns and incompatibilities resulting from updated and new plug-ins. As common Eclipse configurations consist of a plethora of plugins (usually maintained by 3rd party developers), there was a constant need for dealing with updates of particular plug-ins, also in cases when no new plugins where to be installed. Even when developers decided to stick to one version of Eclipse as a stable working environment, APIs to shared tools like the CVS

could change and enforce the update of the related plugins. Furthermore, new tasks did sometimes require new tools and therefore new plug-ins were explored and introduced.

The companies we investigated expressed quite different strategies of dealing with the challenges outlined above. These approaches move along a continuum, whose ends are *autonomy* (leaving the responsibility to configure and maintain Eclipse to the individual developers) and *heteronomy* (managing Eclipse configurations in a centralized manner, thus enforcing a homogeneous working environment for the whole team). Following Max Weber (1949), we conveyed and conceptualized these approaches into three different "ideal types", which represent the underlying strategies of the companies for managing appropriation in practice: the *lone warrior* type, the *centralized organization* type, and the *collegial collaboration* type. In the next sections, we will discuss these types and illustrate their underlying practices as well as their socio-technical implications.

5.4.1. Lone Warrior

The first type is conceptually constituted by a high autonomy of the team members. It refers to the freedom, but also to the individual responsibility workers have for taking care of their workplaces. There are no guidelines for setting up the individual Eclipse configurations, although the limitations of the platform may force team members to use very similar configurations for certain collaborative functions, e.g. if only one plugin is available for connecting to the version control system. If changes to the workplace configuration need to be introduced on the team level, practitioners have to justify this constriction of the individual autonomy by proving the benefits of the suggested change to each of his colleagues.

Empirically, the lone warrior practice is expressed by the circumstances that collaborative appropriation is not the normal case, but occurs at most accidentally (e.g. in response to breakdowns of the infrastructure). In our study, an example for such an accidental collaboration was caused by the breakdown of a shared source code repository that was caused by an update. In this exceptional case, the incompatibility forced the team to search for a solution that was then adopted by the whole workgroup (in the specific case, a developer worked out that the problem could be fixed by using a different Eclipse plug-in. This information was spread and the plug-in was then used by everyone). Except of such rare situations of collective appropriation of new tools, every developer decided on his own what plug-ins to install and how they should be configured.

As the lone warrior culture forced team members to search for useful tools themselves and experiment with them, practitioners were often very knowledgeable about their individual workplace. On the other hand, we recognized a lack of group-consciousness regarding tools and existing tool expertise. For example, in one company no one in the group was able to tell us what tools his colleagues were using. From a management perspective, this lack of awareness can have negative effects, like the tendency to develop heterogeneous and possibly conflicting work practices, decreasing the diffusion of innovation among the team members, and finally making it more difficult to find the right person to ask for advice in case something went wrong or in case team members evaluated new tools (Boden, Draxler & Wulf 2010).

5.4.2. Centralized Organization

This type is constituted by heteronomy that stems from a centralized organization of the Eclipse installation. In this case, the workplace is designed by a designated team member and then distributed to the team. Changes to the common installation have to be discussed with the developer in charge, who is responsible for updating the installation and dealing with the possible incompatibilities.

Empirically, we observed one case of centralized organization as the opposite extreme to the lone warrior type. In this case, the team elected a *tool care taker*. Once a month, his duty was to configure a stable working environment that met the needs of the group and afterwards put it on a shared file system where it was accessible for all team members. He described this work as follows:

"Developer 2: [...] *I sit down at home on my Vista partition and an up-to-date Eclipse [configuration].* [...] *I have a list of plug-ins which should be included. [...] I update this at home, put it on an USB stick and bring it here. Here, I upload it to a shared drive so that everybody can copy it."*(Transcript from an interview with the tool care taker).

The decisions about what should be included in the common configuration were made by the whole team during stand-up meetings. Although the decision process was collaborative, the *workplace caretaker* realized the common working environment on his own. He was expected to be interested and quite skilled in finding new tools as well as in being up-to-date. Team members even reported to respect him as a "primus inter pares" in this regard. Everyone else within the team relied on his skills to regularly set up an up-to-date, stable environment for the whole team.

The approach of centralized organization did not just create the opportunity to benefit from the maintenance of a common and thus homogeneous workplace configuration, but also created a latent obligation for other developers not to customize their work-

places. As experimenting with new tools for personal purposes was not legitimized by the team, the amount of influence team members could bring into discussions about the configuration management was clearly limited. The reason for this was, that it is difficult to say what the concrete benefits are before one has used a new tool in practice. Hence, the central organization type had the tendency to impede individual initiatives of developers to explore new stuff during daily work. It detached innovation activities from the daily production work. This means that the centralized organization type had to solve the latent demand to innovate in different ways than the lone warrior case, which allowed easy exploration through the integration with the daily work, which was now mainly in the responsibility of one central team member.

5.4.3. Collegial Collaboration

As we have outlined in the two previous sections, our observations showed that in practice there were often exceptions from the rules impeded by the different ideal approaches that the companies chose to deal with their workplace configuration. This led to the identification of a third type, which is constituted by the dialectics between autonomy and heteronomy. In this type, responsibilities are not pre-defined, but have to be clarified in a situated manner. As a result, the used working environments were often neither as homogenous as a central organization, nor as heterogeneous as in a lone warrior culture.

Empirically, we found many different practices of sharing configurations and plug-ins in the team in an unplanned and ad-hoc way. These were similar to forms of learning that have been described in the literature as learning in over-the-shoulder situations (Twidale 2005). Generally, sharing was mainly based on a copy-and-adapt installation practice, whereas one user configured and adapted her own Eclipse configuration individually, and afterwards shared this configuration with his colleague(s) in the project.

"I did configure Eclipse a few times. Or rather we configured it more or less together. For example, [within the project] we use the CheckStyle component and some other plug-ins, whose names I forgot, because in fact my colleague did set up the original configuration. And I eventually copied the whole configuration over to my workstation." (Transcript from an interview with a junior developer).

An important type of situation was a kind of initiation rite in which experienced Eclipse users introduced new team members or novice Eclipse users to the Eclipse technology (e.g. if a developer joined the company and/or the team). In such situations, senior developers would often copy their working environments to the machines of the junior developers. One observed effect in such situations was that people

worked with similar configurations for a certain time. Later, the working environments would drift apart in a lone warrior manner, as everyone modified the environment for his own purposes.

The result of such a working style was that the selective cooperation fostered homogenization to a certain degree, but did not enforce it in a centralized manner. At the same time, the related ad hoc collaboration did not result in a systematic diffusion of innovations. Instead, tools and experiences were typically shared in an unsystematic manner, which was also a result of very limited support by existing technology. Accordingly, we found several attempts to organize self-made infrastructures to overcome existing shortcomings in technology support.

"Developer: So, if the project requires to install something new and interesting, [...] I send a request using the mailing list: 'Who knows a good plug-in for that?' Interviewer: What do you mean by 'mailing list'? Who receives these emails? Developer: [...] All employees receive this message and everyone who wants to comment can answer directly. [...] Some time ago, when we really wanted to use [a certain plug-in], I had already heard about it on the mailing list. [...] And I directly contacted the colleague, asking where I could get it and how I could use it." (Transcript from an interview with a developer).

In this case, employees utilized the company's mailing list to reach every colleague and to describe what tools they were currently using in a new project, as well as experiences regarding the usage of tools. This presents a notable example of creating a proxy for over-the-shoulder situations, coping with the challenge that several employees worked at the customer's site full time as project leaders, programmers and technology consultants. In particular, these people described that they felt better connected to their company colleagues because of this mechanism. While they considered themselves as part of the team, they also considered themselves as part of the periphery. Living in the diaspora they were afraid of being cut off from their colleagues at the headquarters, who were perceived as leaders in appropriating domain specific innovations.

5.5. Conclusion

In this paper we studied practices of appropriating software in the Eclipse ecosystem, asking how small teams of software developers manage their working environments. We distinguished between three different approaches of managing the workplace configuration. The first one was labeled the *lone warrior* type. It is defined by the concept of autonomy, while the second, the *centralized organization* type, is defined by

the concept of heteronomy. Both types have in common that informal collaboration among team members is not a constitutive element. This distinguishes them from the third type of *collegial collaboration*, which is defined by *autonomy* and *heteronomy* as a dialectical unity.

It has to be noted, that the three types are analytic categories in the sense of Max Weber's "ideal types" (Weber 1949). In practice, their borders were blurred as we observed various forms of collegial collaboration in all cases, leading to a fined-grained balance between autonomy and heteronomy. Sharing occurred often rather unplanned in over-the-shoulder situations, in case of breakdowns, in situations where newcomers joined the team or just by accident. In these situations, the sharing of configurations and knowledge did lead to a temporal increase in awareness on what is used and homogeneity of configurations. Between such occasions, configurations usually drifted apart as users added new plug-ins or changed configurations to their needs, which led to an increased heterogeneity and a decreased awareness which could sometimes cause breakdowns in the cooperative work. We also observed attempts of overcoming existing shortcomings by realizing collegial forms of appropriation in distributed work. Examples were setting up mailing lists to share experiences about new tools in a more systematic way, or using the source code repository to also store the tools with the code.

Nowadays, most tools provide tailorability functions. Eclipse as our example can be tailored by adding plug-ins, setting preferences and changing the design of the user interface. However, our research showed a clear lack of tools for supporting the various practices of collaboration with regard to appropriating/tailoring. This is especially true for tailoring artifacts in complex software ecosystems like Eclipse, since modifications are usually delivered by 3rd parties. In such cases conditions as practicality, compatibility, stability can hardly be estimated beforehand, making the selection of tools difficult and risky. Therefore, further research is needed to explore how collegial collaboration can be supported with regard to appropriation. Such approaches should not only take into account the provisioning models represented by the three ideal types, but also the dialectical unity of autonomy and heteronomy that framed the different practices we found in the field. In doing so, they should support tool awareness as well as practices of over-the-shoulder learning.

6. Situated Practices of Appropriating the Eclipse IDE[17]

Software engineers and developers are surrounded by highly complex software systems. What does it take to cope with these? We introduce a field study that explores the maintenance of the Eclipse Integrated Development Environment by software developers as part of their daily work. The study focuses on appropriation of the Eclipse IDE. We present an empirical view on appropriation as a means to maintain the collective ability to work. We visited seven different organizations and observed and interviewed their members. Each organization was chosen to provide an overall picture of Eclipse use throughout the industry. The results decompose the appropriation of Eclipse by software developers in organizations into four categories: learning, tailoring and discovering, as well as the cross-cutting category: collaboration. The categories are grounded in situations that provoked a need to change as well as in policies adopted for coping with this need. By discussing these categories against the background of Eclipse and its ecosystem, we want to illustrate in what ways appropriation of component- or plugin- based software is nowadays a common and highly complex challenge for Eclipse users, and how the related appropriation practices can be supported by IT systems.

17. © 2014 IEEE. Reprinted, with permission, from Sebastian Draxler, Gunnar Stevens and Alexander Boden, Keeping the development environment up to date - A Study of the Situated Practices of Appropriating the Eclipse IDE, September 2014. http://dx.doi.org/10.1109/TSE.2014.2354047

This chapter is was published as: Draxler, Sebastian; Stevens, Gunnar; Boden, Alexander; 2014. Keeping the development environment up to date - A Study of the Situated Practices of Appropriating the Eclipse IDE, IEEE Transactions on Software Engineering (TSE), accepted with minor revision.

6.1. Introduction

Adopting appropriate tools and using them effectively has long been recognized as an important factor for improving productivity as well as product quality (Bruckhaus et al. 1996). Therefore, unsurprisingly, the history of software engineering is also deeply entangled with the history of development tools. Back in the early era of computing in the 1950s, development tools were a scarce resource and it was not unusual for programmers to 'hand craft' their own. Today the situation is rather different —an abundance of development tools and other applications. Web browsers such as Chrome or Firefox, for instance, can be individualized by hundreds of extensions, and professional tools like Photoshop benefit from vivid ecosystems of add-on providers. This trend also holds for contemporary development tools. Users of the popular Integrated Development Environment (IDE) Eclipse, for instance, can choose between several different UML modeling tools, code management solutions, database editors, to name but a few. The same is true for other popular development tools such as Visual Studio, Netbeans, SharpDevelop or IDEA.

Tools and tool integration are broadly recognized as a central concern for software engineering. However, we know little about what this abundance of development tools means for developers in practice. Our understanding of the integration work of software developers in real settings is quite limited. Therefore, this paper presents a grounded theoretical study of appropriation work, providing the broader context in which tool integration in companies is embedded and illuminating practices on the 'shop floor'. By providing a rich account of these practices, we will show how appropriation is constituted in learning, tailoring and discovering as well as collaboration. Furthermore, we will show how Eclipse users' actions and their technical, social and organizational situations are intertwined.

Our results are based on an ethnographic field study in seven heterogeneous organizations involved in software development (Table 6). We observed software developers during their daily work, examined their workplace context and interviewed them about their tool usage and appropriation practices. All these organizations, despite their differences, use the Eclipse IDE as their primary development environment. This gave us the chance to perform a comparative analysis to identify both commonalities and differences between the cases.

The paper is structured as follows: in section 6.2, we give an overview of the history of software development tools, as well as their appropriation and integration. In section 6.3, we outline our methodology and the details of the study. Following this, in section 6.4, we discuss details of the Eclipse ecosystem and the features which pertain to ap-

propriation at the shop floor. In section 6.5 we analyze appropriation work in relation to the various dimensions which affect it. Finally, in section 6.6, we discuss our findings with regard to research on software development environments.

6.2. Related Literature

Workplaces where software engineering is done usually combine different tools for computer aided software engineering (CASE) and integrate their features into a powerful and often complex ecology. In the following, we give a brief survey of research into workplace design and discuss why the use-perspective is becoming ever more important.

6.2.1. Designing and integrating CASE tools

The first CASE tools aimed at the automation of routinized, basic tasks (like using compilers to generate machine code) (Kernighan & Plauger 1976) were developed in the 1970s. Since then, significant progress has been made, both qualitatively, as the complexity of task support increased (Bennett et al. 2008) and quantitatively, as the variety and number of tools grew (Forte 1993). The design of CASE tools in this time increasingly oriented towards the paradigm of 'software factories' (Boehm 2006). The guiding principle of software factories included separating planning processes from the work at the shop floor (McClure 1989; Fernstrom, Narfelt & Ohlsson 1992). Here, CASE tools were introduced to ensure that work practices were compliant with prescribed development methodologies and processes (Zettel 2005). As a result, individual developers were largely excluded from the design of their own workplaces.

The latest paradigmatic turn in the design of Software Engineering tools has been to distinguish repetitive software engineering tasks that could be streamlined via automation from the "essential" tasks that rest on human expertise, judgment, and collaboration (Brooks & Jr. 1987). This trend is reflected by the less restrictive CASE tool design associated with Integrated Development Environments (IDE) (Lundell & Lings 2004). In particular, IDE such as Microsoft's Visual Studio and Eclipse present a synthesis of the need for individual tailorability and the need for standardization (Geer 2005): On the one hand, they provide freedom for users to adapt the working environment to their needs (which calls for related organizational protocols (Gamma & Beck 2004)). On the other hand, they contribute to the automation of repetitive tasks as well as to standardization.

The standardization of working environments also progressed, e.g. by the introduction of the NIST/ECMA reference model (NIST/ECMA TR/55 1993) for tool integration.

More recent approaches investigated activity-centered tool integration (Hansen 2003), model based integration (Asplund et al. 2011) and developing measures to determine how well a tool might be integrated (Zelkowitz 1996). A promising approach is furthermore the combination of sharing artifacts and experiences, as both seem interrelated (Bourguin, Lewandowski & Lewkowicz 2013). However, despite that progress, working environments today often consist of a plethora of tools which are only *"partially integrated, forming a complex tool landscape with partial integration." (Asplund et al. 2011).*

6.2.2. Adopting CASE tools

A number of studies on the impact of CASE tools on software development work were carried out in the 1990s. While several studies mentioned positive effects (e.g. (Finlay & Mitchell 1994; Aaen et al. 1992)), others reported that introduced CASE tools were never used (Kemerer 1992) or subsequently abandoned (Elshazly & Gover 1993). To understand the reasons in more detail, several studies on CASE Tool adoption have been conducted.

Focusing on the organizational level, Orlikowski's widely cited work (Orlikowski 1993) showed that the adoption of CASE Tools is complex and dynamic. In addition, she demonstrated that adoption should be understood as an organizational change process that depends as much on organizational and political issues as on technical ones. Maansari and Iivari (1999) further analyzed how expectations and perceptions changed before and after an organization-wide introduction of CASE tools. Several studies (Maansaari & Iivari 1999; Rai & Patnayakuni 1996; Mendoza, Rojas & Pérez 2001) showed how work process compatibility and management support are important factors for adoption. Yet, as Riemenschneider et al. (2002) noted, *"contrary to common beliefs, acceptance of the methodology [and tools] is not assured just because it is mandated by the organization" (Riemenschneider, Hardgrave & Davis 2002).*

In order to understand the relevant factors affecting CASE tool adoption by the individual, research focused mainly on adoption theories like the Technology Acceptance Model (Davis 1989) or the Personal Computing Utilization Model (Thompson, Higgins & Howell 1991). Both show a significant relationship between adoption and factors such as ease of use, perceived usefulness, and long-term consequences (Iivari 1996; Chau 1996). Furthermore, (Kemerer 1992; Fowler et al. 2000) have shown that adoption is linked to changes in work behavior, requiring new skills and knowledge on the part of individual users. Moreover, it was argued (Robbins 2005) that tool adoption can lead to the adoption of new methodologies and vice versa.

Software development is today largely team-based (Riemenschneider, Hardgrave & Davis 2002; Chau 1996; Sharma & Rai 2000; Hardgrave & Johnson 2003) which implies a third important influence on adoption, situated between the individual and the organization. However despite its importance, only few studies explicitly address team practices. Existing studies typically argue that the social context has a double impact: With regard to social norms, teams exert a pressure to accept and internalize common and compatible styles of work. These pressures can promote (Iivari 1996; Sharma & Rai 2000), but also impede the adoption of new tools and methodologies (Riemenschneider, Hardgrave & Davis 2002).

6.2.3. CSCW studies on tool appropriation

The social dimension of software adoption and adaption was investigated in more detail in the field of CSCW. In the 1980s and 90s, studies showed how non-programmers tailored newly introduced systems in the local context of non-SE work environments. MacLean (1990) investigated a system that allowed office workers to create buttons on their user interfaces which contained small macro-like functionality. Furthermore, these small programs could be shared through an email-like system. In a similar way, patterns of cooperation emerging in the use of the sophisticated tailoring and extension facilities offered by many CAD products were identified (Gantt & Nardi 1992). Further, studies on X-Windows modification and email filter configuration identified common patterns of sharing tailored artifacts within an organization (Mackay 1990a). These studies showed that even non-programmers can tailor in a small way. More complex modifications, however, are usually carried out only by more motivated and technically skilled users. Furthermore, both studies (Gantt & Nardi 1992; Mackay 1990a) identified a third group of users, situated between end-users and developers, able to contextualize existing modifications by acting as bridges between developers and users who do not tailor themselves. MacLean et al. (1990), called the three groups programmers, tinkerers, and workers. In Mackay's work (Mackay 1990a), they are called lead users, translators, and ordinary users.

A later study that focused more on collaborative aspects of tailoring (Kahler 2001b) argued for support mechanisms for tailored artifacts. These include repositories for tailored artifacts, supplemented by features for sharing knowledge about how these artifacts should be used.

One of the few studies in the context of modern IDE usage (Murphy-Hill & Murphy 2011) showed that the first problem of adoption is the necessity to discover relevant artifacts. The discovery mode described as *"peer interaction"* between developers includes observations as well as direct recommendations in order to become aware of in-

teresting artifacts for adoption. This can happen face to face or even through the use of eclectic tools like social media and instant messengers in order to share customizations and especially to help peers. This can be considered especially important, as e.g. many Java developers (12 of 14, based on interaction histories during a study of Eclipse usage) use third-party plug-ins (Murphy, Kersten & Findlater 2006).

Furthermore, the benefit of role-based, coarse grained tailoring functionalities in the context of the Eclipse IDE (IBM Rational Application Developer – RAD) have been shown (Findlater, McGrenere & Modjeska 2008). The findings suggested that tailoring mechanisms that are finer grained and more related to tasks instead of roles, as used by RAD, may be more effective.

Furthermore, beneficial to understanding the results of this study is Sigfridsson's endeavor to understand how developers (in distributed projects) gain and develop their practical knowledge (Sigfridsson 2010). Although it does not focus on adoption and adaption, it presents how three categories (continuos change, equivocal situations and external influence) lead to a very complex field of work and at the same time force its members to continuously adapt their practical knowledge.

During recent years, efforts were made to cover both, adaptation (tailoring) and adoption (deciding to use something), under a single term: appropriation. The term relates not only to the repurposing of technical artifacts for one's own aims or the work involved in doing so, but is held to be a general prerequisite for making practical use of technologies (Pipek 2005; Dourish 2003; Balka & Wagner 2006). In this sense, the concept can be used as a guiding principle for field studies on software adaption and adoption.

6.2.4. Discussion

During the last few decades, great progress in the design of SE workplaces has been made, entailing a shift from designing isolated tools to highly tailorable working environments which allow the easy integration of additional tools. In addition, the way tools are distributed and integrated has radically changed. Instead of pre-packaged software bundles and occasional updates, the rise of the internet has fostered the creation of open and standardized platforms, accompanied by product communities. These so called software ecosystems (Bosch & Bosch-Sijtsema 2010a) have radically changed the whole tool market. More 3rd parties (commercial companies, as well as single persons) began to build tools, offering new features and bug fixes in very fast cycles. These tools can be installed by Eclipse users, mostly free of cost and during runtime (Des Rivières & Wiegand 2004). As a result, the integration work has to some extent shifted from plug-in manufacturers to Eclipse users. Truly identical working en-

vironments became increasingly rare (Draxler & Stevens 2011). Sometimes the Eclipse user is the first person who runs an "integration test" that verifies if this particular set of installed tools is compatible.

The appropriation work that emerges with this tailoring of modern tool ecosystems (Bosch & Bosch-Sijtsema 2010a) as part of SE workplaces – has not yet been investigated in detail. From previous research several factors that promote or hinder appropriation are known (Kemerer 1992; Elshazly & Gover 1993; Orlikowski 1993; Maansaari & Iivari 1999; Rai & Patnayakuni 1996; Mendoza, Rojas & Pérez 2001; Riemenschneider, Hardgrave & Davis 2002). In particular: management support, perceived usefulness and ease of use are held to be factors that significantly influence appropriation. In general, we cannot consider workplace design merely as a technical challenge, but also have to understand it as a change process that involves the individual, the organizational and the team level. However, these 'social' factors have not, as yet, been investigated in relation to professional software developers.

In order to contribute to an understanding of tool adoption and adaptation of software developers in its social context, we want to combine the tradition of grounded theoretical studies in Software Engineering (Orlikowski 1993; Coleman & O'Connor 2008) with traditional ethnographical work on collaborative work practices in CSCW (Mackay 1990a; Schmidt 2000) to uncover the underlying structure of appropriation work in the case of modern IDEs. For an example of this approach, applied to research software development work, see (Sigfridsson 2010).

6.3. Research Methodology

For our study, we chose the Grounded Theory approach (Orlikowski 1993; Strauss & Corbin 1997; Shull, Singer & Sjøberg 2008). This allowed us to recover the structure of appropriation work from within the field in an explorative manner. It also allowed a focus on contextualized and situated practices that have been omitted in tool adoption research thus far, and hence the building of substantial theories that are closely tied to the observed context, yet nevertheless are fit for 'analytic generalization' (Yin 2009). Our aim was therefore not to quantify practice occurrences through searching for predefined categories, but to explore the range of existing practices.

The literature on Grounded Theory is shaped by an ongoing debate on the usage of *preconceived categories and existing literature* as material for analysis, going back to slightly different variants of the method's founders, Glaser and Strauss. Our analysis followed the approach of Strauss, which allows the inclusion of existing concepts and

findings from literature during coding phases. All notes, interviews and related artifacts that we gathered in the field, were coded using this approach.

In 2011, parts of our study have been published with a focus on appropriation work at one small company (called Alpha in this publication) (Draxler & Stevens 2011). This paper extends the initial study through including field data of six heterogeneous organizations, leading to important clarifications and reconsiderations in our analytic understanding.

6.3.1. Sensitizing and constitutive concepts

The theoretic concepts of *appropriation*, *practices* and *situations* are of special importance for this field study. They were shaped by our experiences in the field and have proven constitutive of the phenomenon of appropriation work.

In this paper, we understand *appropriation* as the network of activities users continuously perform to use software artifacts constructively and to incorporate software artifacts into their lives for better or for worse (Stevens, Pipek & Wulf 2010; Poole & DeSanctis 1989). We understand *practice* as a routinized pattern of human action (Wulf et al. 2011). It is represented by a ‚skillful performance' of activities, which are reproduced in a specific context. A practice is grounded in, often implicit, background knowledge and implicit normative rules, which render the action meaningful within its context. In contrast we understand a *situation* as a specific constellation in which an activity takes place. Appropriation is therefore mostly a lens for our research and we will undertake efforts to explore its meaning in the context of Eclipse IDE usage. We intend to gain this understanding by investigating observable and describable *practices* and *situations* of Eclipse IDE users.

6.3.2. Data sources, selection and analysis

The data for this study was collected between 2006 and 2009 at seven different research sites (Alpha to Theta). All sites are German organizations which were at the time involved in software development. All of them utilized the Eclipse IDE for at least part of that work. The sites were chosen specifically for the comparison of results (that way Grounded Theory achieves significance). Therefore, we chose organizations of different size, structure, customer base, business models and technology used (see Table 6).

At three sites, we started with several days of work/workplace observations. At every location, we carried out two to four on-site interviews in situ. Most interviews took

around 1.5 hours. Few interviews were shorter in duration and some took about 2 hours. In addition, we collected Eclipse configuration files.

Company	Employees	Domain	Gathered data
Alpha	10	Software Development (shared workspace system)	3 interviews 3 Eclipse footprints
Beta	8 + free-lancers	Software Development (business games, web development)	3 interviews 3 day observation 5 Eclipse footprints (of 4 persons)
Gamma	ca. 1200 (ca. 90 developers)	Insurance company + in-house software dev.	3 interviews 5 Eclipse footprints
Delta	ca. 50	Software Development and consulting (Eclipse RCP, web development)	4 Interviews 2 Eclipse footprints 5 day observation
Epsilon	>2000 (30 maintenance dept.)	Research facility, Inhouse development for Machine maintenance	3 interviews 6 Eclipse footprints
Zeta	ca. 250	Software Development (multimedia and web applications)	5 day observation 4 interviews
Theta	ca. 35	Software development (Eclipse RCP, web development)	2 interviews 3 + 7 Eclipse fooprints of two teams

Table 6: Visited research sites.

6.3.2.1. Observations

At Beta, Delta and Zeta, we carried out several days of observation. At Beta, we tried to observe the whole company because its small size and work structure did not include fixed development teams. At Delta and Zeta,- much bigger organizations- we concentrated on particular development teams. We followed the observed teams closely, took notes and asked questions at appropriate times.

6.3.2.2. Interviews

Overall, we interviewed 21 people at their desks or in meeting rooms of their organizations. Interviews that started in a meeting room always ended with visiting the actual workplace. Therefore, we were able to observe modifications the interviewees made to their Eclipse installation(s). This again enabled us to ask detailed questions about

these observations. We regularly made use of this opportunity to clarify responses given during the interviews.

The participants' Eclipse usage experience ranged from 2 to 9 years. Some were final year university students but most were professional software developers with several years' experience. We also interviewed the CEO of Alpha as well as the CTO of Beta, who both did some development work from time to time. At the larger organizations (e.g. Gamma and Epsilon), we interviewed people who worked together as a workgroup or development team.

Before our field trips, we prepared an interview guideline. It included the following topics:

- Personal aspects such as age, professional career, period of Eclipse usage and experiences.
- Aspects related to the organization such as size, business model, and style of working.
- Detailed questions on Eclipse usage, configuration efforts and collaboration.

Following the iterative and adaptive work-theory building principle of Grounded Theory (Strauss & Corbin 1997), we extended and focused our guideline during the study, based on the results of earlier interviews. Additionally, if participants brought up related topics that seemed to contribute to our research question, we asked further questions to investigate this in more detail. All Interviews were audio-recorded and transcribed with the consent of interviewees.

6.3.2.3. Artifact analysis – Eclipse footprints

The Eclipse development environment is a complex product and Eclipse users can tailor substantial aspects of it. The key questions in our interviews centered around which software components (features/plug-ins) had additionally been installed by an Eclipse user and which preference settings had been changed.

Eclipse allows us to extract this kind of information and export it to a plain text file, naming each of the installed components as well as their version, preference settings etc. These files can be seen as 'footprints' of Eclipse installations. They provide insight into the configuration decisions of Eclipse users. We carefully analyzed the footprints by comparing objective data with the statements of the owner of this Eclipse installation. Even though the extraction mechanism changed between different versions of the Eclipse IDE, which rendered a quantitative analysis less meaningful, the configuration files were important evidence for our study as they allowed us to ask further questions

in case of mismatches between interview and footprint or helped to validate the statements from the interviews.

6.4. Understanding the Three Faces of Eclipse

This section will provide an introduction to the Eclipse IDE, its typical users, and the Eclipse community in general, considered as a software ecosystem. It is intended to provide context for readers who are not familiar with the Eclipse project. It should help the understanding of practices, restrictions and problems that will be presented in the following parts of the paper.

6.4.1. Typical Eclipse Users

The Eclipse IDE (standing for Integrated Development Environment) is a tool-set for software developers. Therefore, the user groups associated with it tend to be software architects, developers and testers and other highly trained specialists. Nevertheless, it puts its users in a difficult situation. On the one hand, it is a powerful and very tailorable set of tools that fits a very broad range of tasks. But oftentimes it also forces users to tailor it to their specific needs. Tailoring the Eclipse IDE is often a very complex task. It is time consuming and can generate error, as incompatibilities between components can appear. Problems range from components not working or disappearing, apparently strange system behavior, to completely broken installations. In fact, tailoring Eclipse can become so complex that even software engineers can easily reach their limits.

6.4.2. An architecture and user interface, designed for flexibility

If the practical act of tailoring Eclipse can be quite complicated, the technical foundations that allow this kind of tailorability are just as elaborate. Eclipse was built for flexibility. Since the release of Eclipse IDE version 3.0 in 2004, everything is part of a component, a so called plug-in, which makes Eclipse stand out from most other software. Usually, flexible systems consist of a monolithic core, supplemented by additional components (see Figure 11 left). Eclipse is a complex framework of interconnecting components, delivering all of the functionality (see Figure 11 right). There is literally no monolithic core or base, just a tiny runtime which loads and executes plug-ins. In Eclipse terms: "*Everything is a plug-in*" (Gamma & Beck 2004). From a purely technical point of view, other software may be similarly structured. What makes Eclipse interesting is the fact that Eclipse users are aware of recently introduced architectural specialties.

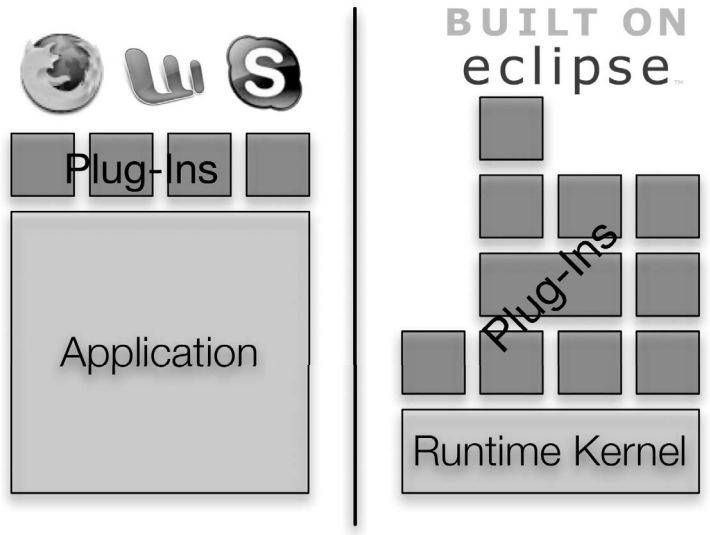

Figure 11: Architectures of traditional software (left) and the „every-thing is a plug-in"-approach of Eclipse (right) as perceived by it's users.

The Eclipse architecture separates components into two main groups. A) Containers, or 'features', used to group functionalities. B) Components that deliver the functionality, called plug-ins and fragments. The Eclipse IDE comes preconfigured with a number of features, and each includes several plug-ins and fragments. Several of such preconfigured packages are offered, all of them specialized Integrated Development Environments (IDE) for software development. Examples are specialized Eclipse IDE for testers, Java developers or C++ developers[18]. Even so, there are often components that are very useful, or even necessary, which are not included. Therefore, an Eclipse user often has to add further features/plug-ins in order to modify Eclipse's behavior to his/her needs. There are over a thousand components available for Eclipse. Theoretically Eclipse users can even write their own components or adapt existing ones to their needs.

Another level of complexity is added as most components are extensively configurable. For example, the text editor component can be configured in terms of font, font size, coding conventions, colors, etc. As an example, Eclipse IDE version 3.6 with additional components for Android mobile phone development shows 139 pages of preference settings, with each page showing several settings. Often these preference settings are preconfigured to be usable right away. Sometimes, however, they need to be configured in order to work properly. A tool that warns the user about bad coding style needs to know what problems it should mark. Similar complex possibilities and necessities for configuration occur with many components.

18. http://www.eclipse.org/downloads/ (Accessed February 24th, 2014)

A third level of complexity is added as the Eclipse user interface is also highly configurable. It is structured into Views, Editors and Perspectives. Views and Editors are small windows within the Eclipse window. They show only a very small and specific portion of information. Examples are a log of warnings and errors, files of a project or a list of bugs to fix. Interacting with the content of a View often activates or opens a different View to show details, or an Editor to interact with and change that information. Views can be modified in a variety of ways. Views can be opened and existing ones can be closed. They can be resized and relocated within the Eclipse window. Furthermore, Views can be minimized, showing the view as and when it is needed, but hiding it if not. A further aspect of Eclipse UI architecture is the concept of Perspectives. A Perspective is a set of Views, with defined locations, sizes and modes. Perspectives are an easy way to switch e.g. between a set of Views used for modeling, and a set of Views for coding or debugging. They reduce the need to open and set-up a dozen views to one click.

6.4.3. The Community's Rhythm

Eclipse is also well known for its lively community (including Eclipse IDE users – ordinary developers and our field for research – as well as people who further develop the Eclipse IDE). This community consists of amateurs, students, freelancers as well as members of companies and large enterprises. The joint effort to make Eclipse continuously better is led by a very small organization: the Eclipse foundation. The Eclipse foundation does not determine the content of an Eclipse installation package or the direction the future development should take. Nor does it do the development work that produces the Eclipse IDE. Rather, it provides connectivity and 'rhythm' for most of the involved people. From 2004 to 2012, the Eclipse community released yearly major version upgrades of the main Eclipse IDE components every June. Some contributors who offer additional components follow this cycle, while others do not. Furthermore, milestone releases as well as unstable nightly builds are available for everyone to use and test. This complicates the situation for Eclipse users even more, as they usually face an upgrade of their main working tool at least once a year (the Eclipse IDE). This is especially complicated, as some components may have changed, while others haven't, introducing new compatibility issues.

6.5. Maintaining the collective ability to work

The Eclipse IDE is a very flexible, powerful and complex tool for software developers. Nevertheless, if changes are necessary, problems can arise. The uncertain rhythm we refer to above means Eclipse users are under pressure to adapt. Therefore *maintaining*

the collective ability to work has become a very important aspect of work. The findings of this study directly address this by presenting the four main aspects of this work to maintain the workplace: learning, tailoring, discovering and collaborating. We introduce these situated work practices by drawing the connections between the need for change and particular situations, problems or issues that were drivers for this need.

6.5.1. Learning and Gathering Information

Throughout the study, we could only find a few restrictions on participants' ability to choose plug-ins and to modify Eclipse for their daily work. When the members of the companies we studied started to use Eclipse, they began to reinstall, modify and improve the software in order to deal with the upcoming challenges of development work as well as new developments in the Eclipse community. The result was a complex and continuous process of gathering information and learning over several years that we present in the following sections.

6.5.1.1. Usual topics to learn about

The software developers we studied used such tools to a significant degree to support their work. To benefit, however, they needed to be aware of what tools at that point existed, what functionality they included, if they were compatible with their environments and how they could be practically used. Therefore, they needed to get an overview of existing tools, i.e. plug-ins, and investigate their functionalities. The first step for participants was to search for a tool that added support for e.g. the source code repository GIT or UML modeling. This was followed by a thorough check to see if that tool was compatible with the currently used Eclipse installation. Several indicators for compatibility checking were used. The most important and easiest aspect to check was the major Eclipse version. But the task could be more complex, as additionally installed plug-ins can be incompatible with each other. Furthermore, plug-ins that were new to the user needed to be thoroughly understood in order to be of practical use. This involved far more effort than getting an overview of the tool's functionality. And lastly, many tools had to be configured in order to be helpful.

6.5.1.2. Learning Practices

A special source of information and a foundation for learning about recently introduced aspects was the Internet. Using general purpose *search engines* such as Google was a typical practice throughout the study. This was a step in a quest for information to solve certain problems or to stay informed of activities in the Eclipse community. Used search terms were the names of known plug-ins, certain topics such

as “PHP programming Eclipse“ or details of a problem encountered by the users (e.g. an error message).

While searching presents the first step, it was just the prelude for *reading*. We found usage of internet-based media such as websites, user communities or mailing lists, which report on software development and Eclipse related topics. This includes news about upcoming or updated plug-ins, configuration manuals as well as tutorials for usage of certain tools and technologies.

The Eclipse magazine is a journal on the German market that exclusively addresses topics around the Eclipse IDE and Eclipse community, for example current plug-ins, their applications and functionalities as well as their configuration possibilities. Even more importantly, perhaps, there are general software development related magazines. These are not focused on Eclipse, but address topics around Eclipse from time to time. Participants read both, although some stated that they did so only from time to time.

Occasionally, tutorials and manuals that explain a certain plug-in refer to an older version of the plug-in. Therefore, learning by doing or *exploring* was considered a very important practice, even when following the existing documentation. When in doubt, the participants explored a plug-in‘s functions, gathered experiences through using it and observed and reflected on their progress in order to assess its benefit for their work. This included creating small example programs to try certain functions. Examples for such tools are code editors for additional programming languages such as Python or PHP, functionalities for the creation of SOAP Web Services, quality control support and source code repositories. Plug-ins under exploration were even used for working on commercial projects, as otherwise it would not have been possible to investigate a new tool or setting in real world settings.

Sometimes, software developers run into problems they cannot solve on their own. A recurrent practice was to turn to a colleague for a different viewpoint or to find possible solutions. These situations offer the possibility *to learn from a colleague's* experiences. During these sessions, the icon of a newly installed plug-in would become visible and trigger a conversation about its functionalities and possible uses. The reason for this help giving activity was often not directly related to tools. However, the study's participants learned about plug-ins and their functions this way.

Learning from colleagues was of course not limited to breakdown situations. During the field study, it became clear that this also happens in a more subtle way. Gamma, Delta as well as Theta from time to time make use of the so called *Pair Programming* technique. Pair programming means two people share one computer during develop-

ment work. One person takes the role of the navigator, while the other person takes the role of the driver. The navigator thinks about the current development task on a larger scale, while the driver thinks more closely about existing code and uses keyboard and mouse to directly manipulate source code. Because of this, they often sit at a colleague's machine. This way, they learn not only about each other's skills, but also about the other person's tools, their configuration as well as their functions. While serendipitous, these practices have several benefits as compared to reading. First of all, certain aspects (e.g. merging branches within a source code repository or the creation of unit tests) can be demonstrated live by the more experienced person in real world customer projects, rather than example code/projects alone. Furthermore, the receiver of the information can assess value more realistically than when information comes from websites, as both persons may have a common background (e.g. working at the same project).

The case of Gamma showed a particularly interesting occurrence of learning, as when two members of that organization told us about a conference visit. The conference addressed general Java-related software development as well as upcoming technologies and tools in that field. Through a talk, they learned about the Mylyn Eclipse plug-in (or Mylar at the time) that was relevant to their work. Their perception was that it would allow them to view their work tasks directly within their Eclipse IDE, instead of using an additional web browser window and furthermore to easily integrate information from their working environment into their task management system. A few days later, they were experimenting with this plug-in and a few weeks later, they begun using it as one of their standard tools. While this is comparable to learning from a colleague, there is an additional layer to it, as they *learned from an unknown person.* Normally, colleagues provide trustable context knowledge but, in this instance, it was the inventor of the plug-ins who was giving the talk, and so his recommendation was trusted.

6.5.1.3. Contrastive Cases

At Alpha, only a few employees were using Eclipse. Senior developers, who were used to working with console based tools of typical Unix/Linux environments, were less likely to see the benefit of Eclipse and refused to constantly struggle with informing themselves. Younger employees, in contrast, had familiarized themselves with the Eclipse IDE during their academic education. They viewed Eclipse as a useful set of tools that was more beneficial than the alternatives. For them, continuous learning about new developments in Eclipse was an integral requirement for the maintenance of their skill levels.

The project team at Gamma, contrastively, delegated almost the whole process of creating a usefully configured, fully working Eclipse installation to one person. Because of this, others told us that reliance on this co-worker meant that reading the usual information sources on Eclipse became less likely.

6.5.2. Tailoring

Tailoring activities were carried out to adapt the Eclipse IDE to the specific needs of its user. These activities were closely related to learning processes, as new needs and new possibilities for tailoring emerged through learning processes.

6.5.2.1. Tailoring possibilities of Eclipse

The simplest form of tailoring was modifying the user interface, using the possibilities introduced in section 6.4.2 The possibility to define perspectives was held to be especially useful and several participants used this to redefine and modify existing perspectives. However, we found no occurrences of newly created perspectives. A more powerful but also more complex tailoring option was the installation of additional components, as these delivered new functionality. Some of these were quite complex and needed to be configured to work properly, which added a new layer of tailoring possibilities. The most powerful tailoring possibility however was to create new plug-ins from scratch.

6.5.2.2. Tailoring Practices

The means to *tailor the user interface* are integrated directly into Eclipse. Perspectives were tailored directly after Eclipse had been installed. This happened e.g. if a new version of Eclipse was released or after a major breakdown of the installation. Additionally, this happened after a new plug-in was installed, as new Views sometimes had to be integrated into a Perspective afterwards. During a work day, participants resized and moved Views regularly if space on the screen was needed for other purposes.

Furthermore, during work place visits and in interviews, we found that people *installed additional plug-ins*. The installation process depended on the form of delivery. At the time of the study, the vendors often offered a download, which usually contained all the files needed. These had to be extracted and manually copied over to a special folder within the Eclipse installation folder. Eclipse would be expected to detect and install the new plug-ins at the next start. This way was considered tricky, however, as it was hard to track whether the installation process worked properly, especially if the user had not previously used the plug-in. If an extension was delivered

through an Update Site, a special user interface could be used to install the extension, the so called Update Manager. After inserting the URL of an Update Site into the Update Manager, it would detect the available Features (which include the plug-ins). The user could select the Features that should be installed and the Update Manager automatically downloaded and installed these. The Update Manager did warn if any errors appeared during the installation but it was not easy for the participants to track down the reported problems. For example, the mechanism complained about missing plug-ins, which were definitely available, downloads stopped in non-reproducible manner or dependencies could not be found, even if the necessary repositories were included. While the Update Manager was the more convenient tool for tailoring Eclipse, the participants stated they used manual installation for a long time. It was only towards the end of the study (around 2009), that users began using the Update Manager more frequently, as it was completely rebuilt and the Eclipse Foundation recommended avoiding manual installation.

Throughout the whole study, we found only one case of custom plug-in development. A member of Alpha *created a plug-in* that delivered code formatting functionality. Compared to most other Eclipse plug-ins, the functionality was quite limited. But in this special situation, incorrectly formatted source code resulted in compiler errors. Therefore it solved a very specific problem for this group.

Eclipse offered a central mechanism *to tailor the configuration of plug-ins*, the so called preference settings dialog. Additionally installed plug-ins often extended this dialog by adding one or more pages with settings that changed the plug-in's behavior. In some cases, additional plug-ins required separate configuration. Similar to plug-ins, preference settings appeared to be quite stable throughout the typical working day in the study. Events that drove the participants to tailor the settings were the installation of new plug-ins, or when Eclipse had to be reinstalled.

The Eclipse IDE is delivered as an archive that can be extracted in any folder of a computer and executed from that place. This enabled users *to install and tailor several instances* of Eclipse in parallel. We found several participants who made use of this feature. They used different versions of Eclipse, e.g. this year's release and last year's release (as required for a certain project). In addition to this behavior, participants kep different Eclipse installations for different tasks. Alpha and Beta both used different programming languages (and related tools) for different projects. Alpha relied on Python, complemented by Java, while Beta used PHP complemented by Java. Employees who often had to switch between projects were likely to use two different Eclipse installations. Each was therefore configured for the project at hand. Participants from Zeta used this technique because of incompatibilities between certain plug-

ins. By using two Eclipse installations, they were able to separate the problematic plug-ins from each other, although this was only possible if the incompatible plug-ins were not both essential for the task.

At Zeta, it was a common practice to archive Eclipse installations and switch to a new one, when a customer-paid project was shipped. This practice allowed the users to unpack and re-use an old, already tailored Eclipse installation that was fully compatible with the shipped software project, even years later. Thus, if the customer required fixes for certain bugs or needed additional functionality, they could start working very quickly.

6.5.2.3. Restrictions on Tailoring

Tailoring activities were sometimes subject to technical or organizational restrictions. These were mandated either by other members or departments of the organization or defined by internal, often implicit agreements. Participants (especially members of Zeta) argued that they felt their superiors expected them to be compatible with other parts of the company, even if this was never really made explicit. Zeta employees explained that while conforming to this expectation, they at the same time tried to explore new and probably unstable Eclipse-related tools from time to time.

Organizational restrictions emerged mainly from daily work and were of an implicit nature. For example, the participants could freely choose additional plug-ins, but felt they had to respect organizational restrictions such as the company's infrastructure or the programming languages in use for the project. While these aspects were clearly external restrictions, the participants argued this imposed no particular constraint since the usage of that particular programming language or infrastructure had been their own (collaborative) decision in the past.

Technical restrictions could be observed in several cases. At Zeta, the usage of plug-ins for web service creation caused breakdowns of the Eclipse IDE. The participants used multiple differently tailored Eclipse installations and backup copies of the whole IDE to solve this problem.

Gamma, as a large company had its own IT department, which used a very restrictively configured Firewall. As a result, the members of Gamma were not able to install additional plug-ins or even updates for Eclipse at all. To avert conflict with the IT department, the team decided to delegate all tailoring tasks to one person. The person most interested in news concerning Eclipse technology tailored an Eclipse installation suitable for the whole team from home and distributed it via a memory stick. This was repeated once a month to make use of updates and integrate additional tools.

6.5.3. Discovering

During the study, it became clear that serendipitous discoveries were an important aspect of actions to maintain the collective ability to work. We found a mixture of discoveries, learning, tailoring and collaboration. Discovering relates to information on tools/plug-ins, their functionalities, compatibility issues, etc. Discoveries can trigger individual or collaborative learning or tailoring activities. Tailoring activities can, however, also result in new discoveries.

6.5.3.1. Things to Discover

The different groups' work practices were closely related to certain plug-ins, as the functionalities and usability of each plug-in suggested a certain style of working. For example, the usage of a code editor for the Python programming language indicated Python development. Discoveries of new and helpful Eclipse plug-ins were therefore important events. These discoveries had the most potential to change future work practices. At Alpha, the Eclipse plug-ins for Python development were spread entirely by serendipitous discoveries.

As Eclipse plug-ins tend to be complex, the settings of a given plug-in were often arrived at through discovery. This happened especially with plug-ins for quality control. FindBugs and Checkstyle are two plug-ins in this category. They made extensive use of tailorable rules to display warnings related to the code the users wrote. As these were only of interest in combination with the rules in use, these settings were very important.

Single functions or even the keyboard shortcuts of plug-ins were also subject to discoveries. It was reported that new icons on someone else's screen led to certain unknown but potentially interesting new functions (e.g. the Eclemma plug-in for unit test coverage support adds one new icon to the toolbar which is visually very different).

6.5.3.2. Serendipitious Situations that Lead to Discoveries

At Gamma, the above-mentioned *conference visit* led to the discovery of a new plug-in (Mylyn). After returning to the company, information seeking and learning practices were used to explore the benefits of the Mylyn plug-in for the project team. *Reading* the existing print magazines led also to discoveries. As with the conference visit, the participants' practice of reading magazine articles on Eclipse led to discoveries and contributed to learning about new plug-ins.

Similar conclusions can be drawn for online magazines. While most of these report on software engineering in general, they sporadically also report on Eclipse. New plug-ins may only appear once a month or every few months, but even reading a note about the update of an existing plug-in might be a discovery.

Practices that led to beneficial discoveries involved personal contact with colleagues. Examples were *collaboratively solving problems or working together* at one machine. These practices sometimes revealed new icons and keyboard shortcuts that led to previously unknown possibilities (e.g. a shortcut to organize imported java packages or reformatting the source code). Learning and tailoring processes followed these discoveries, as with practices reported above. Distinct from these, however, personal contact was perceived in this instance as especially valuable. Participants argued that third party information sources are often very interesting, but co-workers share similar work contexts. They work collaboratively on the same projects, use the same company infrastructure and can rely on common success and failure stories when discussing the use and limits of a certain tool. This rendered discoveries at a co-worker's workplace more useful and trustworthy than discoveries made through magazines or online sources.

The usage of *exploration environments* also led to discoveries. Exploration environments were mostly freshly released Eclipse versions which were tailored in order to explore new functionalities or to test for compatibility issues. During this exploration process, participants sometimes expected to make discoveries (e.g. to find a new version of a plug-in compatible to the project, while the previous version might have been incompatible). We were also told that this process had been accompanied by unexpected discoveries in the past, as newly introduced functionalities appeared at the user interface. E.g. at the time of the study, support for the Subversion source repository was included with the default download packages of the Eclipse IDE. Prior to this, Eclipse users had to get to know about this plug-ins in other ways.

6.5.4. Collaboration

The problem of maintaining the ability to work over a longer period concerns the whole group, as every team member can run into problems and people from time to time depend on each other's work. A variety of actions that contribute to appropriation, could also be seen as collaboration, including sharing information and virtual artifacts as well as asking for help and solving problems together.

6.5.4.1. Actors and Areas for Collaboration

Collaborative effects could be observed first and foremost between *colleagues*, working together as a group or project team or as members of an organization. Even between co-workers, different levels and areas of expertise became visible. The perceived level of expertise of a colleague was implicitly used as a measure for trust for this person's recommendations. As part of project work at *the customer's site*, people employed by the customer were also part of this process (therefore the group of collaborators spanned multiple companies). Such individuals by definition knew the customer infrastructure much better than suppliers, and shared knowledge to foster collaborative work with external people.

Several interviewees maintained connections to former co-workers or friends working in the same industry and using similar tools. Additionally, they made use of their *personal social network* to ask for help, to help others to solve problems or to generally discuss tools and related topics. If a problem could not be solved within the organization, they would turn to this circle. This was not confined to work and was also a topic of conversation during leisure time.

The last resort in terms of getting help was to contact the *vendor or developer of a plug-in*. Several participants argued that they would directly contact the vendor if an important plug-in would not work as expected, either in order to get help or to better assess the tool for the given task. We were told of only two occasions where such contacts actually happened. Given that this spans a tool usage of several years, it seems, on the face of it, to be a surprisingly rare event. We have to bear in mind that the Eclipse community is huge and incompatibilities, usage problems, etc. as well as workarounds or warnings are published very quickly in related forums. This explains why contact with the vendor of a plug-ins rare.

6.5.4.2. Collaboration Practices

A frequently observed and described collaboration practice was *asking others for help*. Throughout the study, certain events sparked questions. Examples were breakdowns, as well as unclear or difficult aspects of the tasks at hand. The discussion often also included tools and configuration aspects, e.g. missing rules of quality control tools. Whenever a solution for a certain problem was known within the group, a course of action could be recommended.

During collaborative software development, developers relied on the hardware and software infrastructure in use. Therefore, the configuration settings of plug-ins may be

as important as the installed set of plug-ins itself. A plug-in may be useless if not configured properly in relation to the organization's infrastructure (e.g. source code repository tools), the project's needs (e.g. quality control tools) or informal agreements (version of the interpreter of a programming language). Throughout the study, configuration settings were *spread by word of mouth* or read to a person over the phone. In some more complex settings or when organizational boundaries were crossed, they were exported, *sent by e-mail or instant messenger and imported* into another Eclipse installation. This gave the receiver an exact copy of the configuration settings, which saved time and reduced typing errors.

Furthermore, participants directly *shared plug-ins*, e.g. web service support tools at Zeta, support for PHP development at Beta, support for Python development at Alpha. This was carried out by either sharing the files that represented the particular Eclipse plug-ins or by sharing the URL of the plug-in's update site (see Update Manager in section 6.4.2). That said, extracting and sharing plug-ins from an existing installation was extremely complicated and error prone, as the users would have to manually figure out which set of plug-ins belonged to a certain tool and what the dependencies between plug-ins might be. As a result, the participants only shared files if they kept the downloaded archives that contained the plug-ins. Otherwise, they preferred to *share the complete Eclipse installation*. The latter case was carefully applied to enable the receiver to start working without further configuration needs. This was especially useful when new members joined the company (e.g. a trainee who had just joined Beta at the time) or the project team (which happened regularly at Zeta).

While these sharing practices resulted in manual labor, there were certain *support tools* emerging at the time. Services as Yoxos[19] or Pulse[20] allow their users to create tailored, fully functional Eclipse installations. With a few clicks, users can select the plug-ins that should be included. Nevertheless, only Theta made use of one of these services. The participants argued that they did not know the services well enough to see sufficient benefit from their usage. Furthermore, some of the more interesting features were not free of cost and their use would have involved the organization's management in financial deliberations. Both services also tended to apply an IT-department centric view, as they allowed only the owner of the shared installation to reconfigure the installed plugins, which restricted other users of the installation.

19. https://yoxos.eclipsesource.com/ (Accessed February 24th, 2014)

20. http://www.poweredbypulse.com/ (Accessed February 24th, 2014)

The people at Delta and Theta regularly held meetings. Apart from the often more pressing issues of daily business, they also used these *meetings* to discuss work practices as well as tools in use and their configuration. These meetings were an accepted way for members to share their know-how about tools within the company. This created a certain awareness of what others were using during their daily work and who was experienced in which fields. All companies involved in the study carried out regular meetings and from time to time, all groups discussed the tools in use or tools for future use. But Delta and Theta put considerably more effort into this and, as a result, were more aware of these aspects.

A slightly more top down way of sharing tool-related knowledge happened at Gamma. The *project leader* of the group we investigated was in fact an employee of Delta. He joined the project team at Gamma two years previously, along with some of his colleagues from Delta. At that time, people at Gamma had no experience at all in using the Eclipse IDE and related plug-ins. When the project leader and several of his colleagues joined Gamma, he introduced several new tools as well as practices that he was familiar with (Scrum, pair programming, test driven development). The employees at Gamma accepted these innovations and, as they had never had contact with these tools and practices before, this lead to ongoing learning and tailoring activities.

6.6. Discussion

The results of the field study can be interpreted in two ways. First of all, we want to discuss the practices that we have identified in our fieldwork. Secondly, we want to show potential areas for supporting appropriation, based on the results of the field study.

6.6.1. Configuration and adaptation practices

In our study, we have seen that learning, tailoring, discovering and collaborating were the main areas that contributed to the appropriation of the Eclipse IDE.

Strübing (1992) argues that always moving along the *"bleeding edge of technology"* (Strübing 1992) is part of the habitus of software developers. Sigfridsson mainly sees continuos change, equivocal situations and external influence responsible for this effect of continuos change of practical knowledge (Sigfridsson 2010). In the case of the Eclipse IDE, this edge is defined by what happens in the community and at the different plug-in vendors' sites. It is fast moving and dynamic (similar effects have been described for the PyPy open source community (Sigfridsson 2010)). Changing work tasks provide further justification for learning; or in some cases create a pressing need to

learn (see also (Sigfridsson 2010)) about news on the plug-in market and to stay on top of things (see also (Murphy-Hill & Murphy 2011)). Based on the results of the study, we agree with previous studies (Kemerer 1992) (Fowler et al. 2000) (Robbins 2005) (Murphy-Hill & Murphy 2011) (Sigfridsson 2010), which argue that changes in work behavior stimulate learning or the need for new tools which, in turn, can stimulate new skills. The lively Eclipse ecosystem creates plenty of opportunities, as new tools are constantly created.

Adapting the user interface of Eclipse usually took only seconds and helped the Eclipse users in our study to get a better overview, to use different tools and to be better prepared overall for the task at hand. Other tailoring decisions, such as keeping several differently configured Eclipse configurations, were carried out to improve the performance of the Eclipse IDE and to provide backward compatibility, which has not been described by other authors yet. Mostly, however, Eclipse was tailored in order to use new techniques or work practices, as when using a different programming language, using code management tools or testing tools that might result in either faster development or better quality results. This result is in line with previous research efforts on learning and tailoring in SE environments (see e.g. (Murphy-Hill & Murphy 2011; Findlater, McGrenere & Modjeska 2008; Sigfridsson 2010)).

Discoveries were closely related to awareness of colleagues' workplaces. The flexibility of Eclipse as a toolbox for software developers sometimes meant poor awareness of what other colleagues were using. Discoveries were important, because they could help to satisfy a current need or help in fixing a pressing problem. Most of the employees at Alpha, Beta, Epsilon and Zeta were only aware of a minimal set of Eclipse plug-ins or tools that were in use by their colleagues, which made them quite open for discoveries. At Gamma, Delta and Theta, however, this was different. The employees of Gamma could fairly precisely tell us which plug-ins their colleagues were using and even how these plug-ins were configured. This was because their firewall drove them to delegate all tailoring work to a single person and tailoring decisions were communicated within the team. At Delta and Theta, in contrast, tailoring was an individual effort. Even so, both companies were using pair programming techniques, which regularly made pairs of developers to sit down and use one machine together. This raised awareness of the tools in use. While Gamma was also using pair programming techniques, we found the technical infrastructure was the main reason for this level of awareness. Overall, a constant awareness helped members to stay on top of things, but in groups that had low awareness of each other's workplaces, discoveries played a much bigger role. Mutual awareness of this kind is a cornerstone for helping each other out, implemented in a variety of ways, and a very important practice among devel-

opers. This was previously noted by (Murphy-Hill & Murphy 2011), who suggested further organizational support for such practices; (Sigfridsson 2010), who discussed it as part of practical learning and (Twidale 2005), where it is however mainly discussed as personal help giving.

If we focus on the collaborative aspects that contribute to appropriation, we can identify certain sharing practices that contribute to work effectiveness. This kind of expertise and artifact sharing happened above all when new co-workers joined the company or the project team. This was because new colleagues were expected to contribute to the project quickly, requiring useful (and compatible) tools. Similar effects were described by (Sigfridsson 2010) when developers joined the PyPy open source community.

Overall, we can see that the Eclipse users of our study modified their systems to maintain their collective ability to work. In line with (Sigfridsson 2010), we argue that they did whatever was possible to achieve this task. The related practices were often collaboratively carried out and can be categorized as learning, tailoring, sharing. They were often spawned spontaneously by discoveries that also contributed to the teams' overall workplace awareness.

6.6.2. Potential areas for support and process improvements

As the study has shown, at the time, Eclipse was an example of a very powerful, but also complex and in some situations unstable software system. Improvements to the current Eclipse architecture may be of help. In particular, mechanisms that might prevent plug-ins from harming each other could prove very helpful. However, future solutions that aim at a high configurability or modifiability could also learn from existing practices of Eclipse users and include features for supporting appropriation. Some of the suggested measures have already been implemented (see e.g. (Bourguin, Lewandowski & Lewkowicz 2013; Findlater, McGrenere & Modjeska 2008; Draxler et al. 2012)).

6.6.2.1. Fostering Collaboration

As shown, the introduced practices that contribute to appropriation have very often been collaborative. The sharing of experience through recommendations, demonstrations and collaborative problem solving, as well as sharing of artifacts and settings were present to a large degree in all the participating organizations. However, as the tasks at hand relied heavily on the local context, e.g. the infrastructures of the companies and the projects, people from the team were the main points of contact. Systems

that accommodate easy appropriation should leverage this. One successful example was the possibility of sharing preference settings. While that feature was not available for all settings and not at all for plug-ins, it was used on several occasions. It could be easily improved by connecting Eclipse installations to a network infrastructure in order to implement additional features for sharing installed plug-ins, as well as settings, with colleagues.

6.6.2.2. Making Use of Trust Relations

Such functionality would not only increase the usability of the current Update Manager and settings dialogs, but would also enhance trust between the members of an organization. Co-workers often rely on common success and failure stories. They know their own organizations' infrastructure, as well as their products, best. Throughout our study, it was apparent that participants in the main did know who was responsible for certain aspects of this infrastructure and therefore knew whom to ask in case of breakdown. A network to share artifacts within the organization as suggested by (Bourguin, Lewandowski & Lewkowicz 2013) and (Draxler et al. 2012) could benefit from this, as it would make use of known sources from within the organization, and as a result might create awareness of the tailoring efforts of experienced Eclipse users. This could generate serendipitous discoveries and offer a means to share the object of interest in an easy manner. However, this should also be supplemented by organizational measures, as suggested by (Murphy-Hill & Murphy 2011).

6.6.2.3. Searching and Sharing

We learned that Eclipse users regularly search the web for information on tools, as well as for subjective experiences with plug-ins and their settings. We also learned that the relationship of trust between an Eclipse user and unknown people on the web is different to the relationship of trust between an Eclipse user and his/her colleague. The Eclipse forum and its members constitute a huge and complex network and influence at that level is difficult to manage. The local network of experienced persons is more manageable. It should be possible to at least render the areas of co-workers' expertise visible to, and searchable by, other members of the team in order to provide new possibilities e.g. for people who are experienced in using a certain tool.

6.6.2.4. Creating Workplace-Awareness

Making the expertise of co-workers visible and searchable can help an Eclipse user if he needs a contact person, e.g. because a problem has occurred. But this information could also be used to increase the general awareness of colleagues' workplaces in the

group. Serendipitous discoveries were perceived to be very helpful, but happened very rarely and as the name suggests, by chance. Features that not only keep track of the Eclipse user's experiences, but that announce some features thereof within the group, could create a much more consistent discovery experience (Bourguin, Lewandowski & Lewkowicz 2013; Draxler et al. 2012). In much the same way, similar to such a feature, certain practices like regular meetings to discuss the group's workplaces or pair programming can make discoveries much more likely (Murphy-Hill & Murphy 2011).

6.6.3. Limitations

We presented results from seven German companies that were gained through a qualitative study. While these results are very detailed, there are limitations to the validity of this study.

The results are not easily transferable to other cases. This is true for several dimensions of this study. Without further investigation, the results can not be easily applied to other software ecosystems (e.g. Firefox or Android), other user groups (e.g. casual computer users), as well as other usage scenarios (e.g. private/home usage).

Furthermore, we chose to focus mostly on small and medium sized organizations, as they are very important for the German software industry. However, we are aware of (especially large) companies that do not even allow their employees to tinker with their work places. It is also likely that e.g. sharing of experiences is dealt with quite differently in other cultures (the study included only German companies) and work situations (e.g. offshoring scenarios).

This study is further limited by the fact that we may not have taken "preference settings" to be as important as we might have. This, too, could be a topic for future investigation.

We observed that some interview participants found it difficult to remember tailoring efforts they had initiated in the past. One reason for this might be that those practices are only seldom used. In several cases, we gave examples for what we meant or visited work places as part of the interview, in order to spark the discussion.

6.7. Conclusion

This paper addressed the topic of SE tool appropriation as a means to maintaining a collective ability to work. In doing so, we took a CSCW perspective, focusing on the micro-level of appropriation in the context of Eclipse users' work practices. The focus at micro level certainly allowed us to obtain a detailed understanding of appropriation practices in the context of using software ecosystems, and introduced a perspective on

what Eclipse users actually do. Based on interviews and observations, we provide an interpretation of the reasons behind the work being performed. Generally, the results confirm statements from previous studies that have investigated how users engage with software products and share modifications as well as useful practices with each other (Gantt & Nardi 1992; Mackay 1990a; Sigfridsson 2010). The importance of serendipitous situations has however only partially been discussed (see (Murphy-Hill & Murphy 2011)). As such, we believe that our findings contribute to the state of the art of a) understanding software development groups' practices of maintaining the collective ability to work and b) designing supportive tools for software developers, which nowadays have to be understood as ecosystems rather than as standalone applications. Our study can be seen as an attempt to better understand the needs of Eclipse users and to identify approaches to supporting them in maintaining their ability to work individually as well as in the context of their teams, companies and communities. A broader focus on surrounding technologies related to software development (e.g. programming languages, frameworks, bug trackers) could reveal that these results might be applicable to this domain of work (cf. (Sigfridsson 2010)), and constitute a topic for future research.

With regard to possible support systems, we have shown that software developers in the observed organizations possess a great deal of knowledge about their tools and most of them constantly take action to further improve that (cf. (Sigfridsson 2010)). Within all the visited sites, software developers modified their working environments on their own, in order to adapt these to the tasks at hand. In the observed environments, this non-/self-organized approach was a very helpful and powerful mechanism that worked well and could probably be enhanced by novel support tools that render the related learning and tailoring efforts more visible and accessible to other Eclipse users (see (Bourguin, Lewandowski & Lewkowicz 2013; Draxler et al. 2012; Murphy-Hill & Murphy 2011)).

While the observed style of working seems to be very interesting especially for small and medium-sized companies, this level of individualized tailoring also marks an inherent problem of software ecosystems in general: to provide maximum flexibility in the choice of tools and high levels of stability in the working environment at the same time. How these mismatches can be mitigated, and to what extent companies that rely on more standardized work environments can benefit from the flexible practices we have observed is as yet an open question that we plan to pursue in our future work.

6.8. Acknowledgment

The authors wish to thank the organizations that participated in the study. This endeavor was funded by the German Federal Ministry for Education and Research within the research initiative *Software Engineering 2006*. We thank Dave Randall for his valuable support.

7. Managing Software Portfolios: A Comparative Study[21]

Software applications that can be changed, modified and extended are nowadays pretty mainstream. But only few researchers focused on the role of the users social network for actual modifying practices and hurdles. Therefore this paper, studies in a comparative manner, how users modify software applications by using markets of existing components. We examine two popular applications: the universal tool platform Eclipse as an example for work applications and the game of World of Warcraft as an example for leisure applications. Despite the difference of the contexts, we found common patterns in collaborative actions within the social networks, that lead us to discuss the role of sharing and support for modification awareness for end users.

21. This chapter has been published as: Draxler, Sebastian; Jung, Adrian; Stevens, Gunnar; 2011. Managing software portfolios: A comparative Study. In End-User Development. Third International Symposium, IS-EUD 2011, Torre Canne, Italy, June 7-10, 2011, Proceedings. Lecture Notes in Computer Science. Torre Canne, Italy: Springer. With kind permission from Springer Science and Business Media. http://link.springer.com/chapter/10.1007%2F978-3-642-21530-8_36

7.1. Introduction

For a long time, research on managing software portfolios primarily focused on the appropriation of single applications. At a time, when applications had a clear border and when the software market was very limited this was well-suited. Examples are the work of Mackay (1990a) or Gantt and Nardi (Gantt & Nardi 1992) who empirically investigated into tailoring efforts. One remarkable result of both studies was, how much collaboration in form of artifact or knowledge sharing they had observed. But since then the basic conditions have changed. Today it is often tried to establish so called *software ecosystems* (see section 2.4). They consist of an open, extensible software platform that attracts different manufacturers and hobbyists, creating small-scale components, which can be individually assembled by end users (Bosch 2009). Software ecosystems are an interesting topic to look upon, when it comes to study end user development. They empower the end user to choose or add functionality to their software by orchestrating pre-existing modifications. Compared to the situation of the 1990s, software ecosystems and the involved stakeholders are globally networked. We believe that this changes the users opportunities for modifying applications, since under these circumstances local networks of users (e.g. a company) collaboratively makes use of software ecosystems. Our first goal is therefore to understand: *"How and based on what information do people modify their personal software installations?"* To answer this question, we followed a similar approach as Mackay (Mackay 1990a), which can be described as a set of empirical field studies, consisting of observations and interviews. Since we are not expecting an universal answer, we chose two quite different software ecosystems to investigate into, hoping for contrasting results. First we analyzed how professional software developers modify their Eclipse installations during their day to day work. This was followed by a second series of investigations, that examined how players of the online role playing game World of Warcraft (WoW) modify their game clients during leisure time.

7.2. Two field studies on managing software portfolios

Our research process is loosely oriented on Mackay's (1990a) studies. For each study (Eclipse and WoW) we started exploring the relevant literature on the software, organizations and communities that are related to the case. This was followed by partially-structured interviews and in the case of Eclipse, on-site observations. The work is still carried out as open ended qualitative study. The material presented here was transcribed and the transcripts analyzed using coding mechanisms of the Grounded Theory (Glaser & Strauss 1967) approach. While this is not a full grounded theory (so

far we rely on In-Vivo codes), the approach has still proven very helpful to carefully analyze the material, without subsuming our observations under pre-defined categories from literature.

For the Eclipse case we cooperated with six software companies with 10 to 250 employees. At each company, we conducted at least two semi-structured interviews of at least one hour (altogether, we conducted 17 interviews. Additionally, we visited two of the smaller companies over a period of 3-5 days for on-site observation. For the WoW study, we interviewed a small WoW guild that is constituted by 8 to 10 active members of various game experience and with a different educational background. Whereby a guild is an in-game association of player characters formed to make the accomplishment of group-related tasks easier. We conducted at least two semi-structured interviews of 15 to 60 minutes with each player. Additionally, we recorded the changes in their addon configuration over a period of one month.

7.2.1. Customizing the Eclipse IDE (Study 1)

Eclipse is a multi-language integrated development environment (IDE) and an extensible software ecosystem. It began as a toolbox for the Java programming language at IBM. It was designed to integrate future tools under one roof, using a plug-in mechanism. The platform was made freely available, open source and steered by a non-profit foundation to attract other companies and other contributors.

Eclipse provides all of its functionality on top of a core runtime system and can be extended by using additional third-party plug-ins. The Eclipse core and most additional plug-ins come free of charge and are released under the terms of an open source license. An Eclipse plug-in is constituted by an XML description and Java code that supplies the functionality. There are about 900 tools available that consist of thousands of *plug-ins* called components. Tools are either installed using the included install and update mechanism or just copied to a folder manually.

Software developers as we interviewed and observed have to fulfill quite different tasks, from documenting requirements, modeling, coding, testing, debugging to talking to customers. For many of these tasks special tools are either needed or at least are a great help. Furthermore the resulting artifacts are often shared among other developers who work on the same or similar tasks. Eclipse allows to be extended by additional plug-ins that could support the task in question. Most of the observed and interviewed users were capable of creating such plug-ins on their own. But due to the amount of plug-ins that already exist on the global Eclipse ecosystem and the time and effort

necessary to create a new plug-in, they chose to first search for existing alternatives that might fit their requirements.

One of the key findings of the study, when the need for a new tool arose, suitable recommendations regarding tool selection, installation, and configuration were sought out from co-workers who also found themselves in similar working contexts. We especially could observe this in environments where software developers organized themselves in agile teams. People did trust in their co-workers advice much more than in recommendations found on the Internet or in magazines. Within the observed companies, several related strategies were established or put into practice by accident.

If it was obvious that someone should be told about what plug-ins to install, e.g. if a new person joined a project team, either the plug-in names or even the whole set of artifacts were passed to that person. In case of a problem, people went to colleagues and just asked for advice which plug-in to pick or how to proceed if problems occurred. Unfortunately quite often it was not clear who could be an experienced colleague for a certain topic. As some Eclipse users were constantly trying to stay informed on plug-in related topics, they sometimes stumbled upon interesting news that could also be relevant for their colleagues. This was an interesting source for innovation for their colleagues. But as they did not consider chatting about new tools as a central part of their job, it was often unclear if news were important enough to be shared. On several occasions Eclipse users sat together at one machine to discuss a problem or how to proceed with a project or task. While doing so, the colleague did discover new icons in the Eclipse toolbar and so the topic moved to new plug-ins. The result was an exchange on new interesting plug-ins.

Overall we observed that Eclipse users tended to ensure personal information exchange on which plug-ins are interesting, how to install them and which problems can occur. On the other hand it was often unclear if general plug-in related news, tips, experiences should be spread and through which channels. Therefore this was often triggered by accident.

7.2.2. Customizing World of Warcraft (Study 2)

With more than 12 million subscribers the massive multiplayer online role-playing game World of Warcraft (WoW) is currently the world's largest game of this kind. WoW was developed by Blizzard Entertainment and released in November 2004. Three expansions for the game have been released since then, in addition updates of the game client are regularly released.

WoW is an interesting example for EUD in games, because it is possible for players to create their own add-ons for the game client that is used to play. Add-ons are constituted by describing metadata, XML documents describing the user interface and the functionality, which is written in the scrip language LUA. The development of add-ons is officially encouraged by providing the user with access to certain game API functions. There are currently over 5900 user-created add-ons (Pipek & Kahler 2006) for the player to choose from. Addons are installed by downloading a code package from the internet and then placing it in a specific addon folder in the WoW installation.

The game is designed in a way that people work together in groups to accomplish complex tasks in the game world. Players organize themselves in guilds in order to simplify group building and represent social networks. Although the game is designed in a way that players do not necessarily need addons, they play an important role because they can enhance the players or groups performance. Addon functionality can range from displaying additional information that is helpful to the player to automating certain tasks or reorganizing the built-in chat function. Not every player benefits from certain addons as the usefulness of an addon depends on the role that the player seeks to fulfill in the game.

Throughout the whole interview study, every person had modified his/her game client using addons, although this was not a selection criterion. Most of the collaborative innovation process happens via in game chat and voice chat. All interviewees use an external tool called TeamSpeak to coordinate group-related tasks and having general discussions. TeamSpeak works similar to a Skype or a telephone conference where the players connect to a persistent server in order to talk to each other. One of the key findings of our study was that most of the addons are installed, based on recommendations from other guild members. These recommendations are seen as more or equally important as recommendations on the addon-related websites. The experienced players try to stay up-to-date by informing themselves on various WoW addon sites about updates for their current addons or new addons that could enhance their playing experience. As part of the study, we discovered several non-formal modifications practices.

If players want to accomplish certain tasks or face problems concerning certain game elements, this was often discussed with guild members, as helping other players is quite common. As part of this, players often received recommendations on what addons could simplify the completion of this task. If a problem with an addon or the game client arose, the other guild members were always asked first. Unfortunately, often it was not clear which guild member could be an expert for a certain addon or problem. The continuous use of voice chat benefits the virtual collaboration in a way

that it can create virtual over-the-shoulder learning situations. For example in one specific case a player mentioned certain statistics about his character and another player asked where he could find this statistic. The first player then realized that this statistic was generated by an addon and recommended it. All interviewees use a third party tool, called Curse Client[22] to install new and update existing addons. It was developed to help players in installing new addons and managing their current addon configuration, by providing them with an easy-to-use interface representation of a rich addon database. Information about new and interesting addons is spread verbally to players which are suspected to have an interest in these. New players usually get recommendations on certain addons they should install. These recommendations are usually given to them before they engage a difficult task with other guild members.

7.2.3. Discussion

A common phenomenon that we expected was that people tend to employ pre-existing modifications rather then developing their own, hence the fact that certain people in both studies had the skills to develop them. The interviewees in both studies argued, that using pre-existing modifications saves them a lot of time. What we did not expect were the similarities at the practice level as well as the reasons behind certain actions.

But by comparing the two studies, we found more similarities. Despite the differences of the contexts (work vs. play) and heterogeneous user groups (very skilled vs. varying) the sharing behavior was very similar. In both cases people relied on the recommendations of their friends or co-workers more or as much as they relied on recommendations made by external people or websites. We classified these as follows:

#1 asking, because of a problem

People actively asked their friends or co-workers about their software configuration and which modifications they use. This happened mostly if a problem or a new and unknown task appeared. In both studies we traced this back to the belief that the co-workers or friends better understand each others context and therefore are a more reliable source of information.

22. Curse.com World of Warcraft Addons. http://wow.curse.com/downloads/wow-addons/default.aspx. (last accessed 2011/03/14)

#2 asking, triggered by accident

In several cases, people either accidently observed or discussed the use of an modification unknown to others. This triggered a need for awareness of what the colleague or friend is using and resulted often in a discussion to exchange experience on modifications.

#3 actively spreading the news

If people were actively informing themselves or by accident picking up news on modifications, they tried to spread this information further to (potentially) interested people in their near social network.

#4 actively introducing new workers/players

Sometimes a new person joins the game of WoW or a colleague joins a certain development project or even the company. In this case people did introduce the "junior" not only to the work/game, but also recommended certain modifications.

If we mirror these categories back to the several observations and interviews, they lead back to a lack of awareness within social networks as colleagues or guilds. Even more, there is a constant and latent need for this kind modification or EUD awareness. But since there was no support, it took certain points of interaction like a breakdown, beneficial accident or a complex task to bring the topic of modifications to the center of attention and create this sort of awareness.

7.3. Related work

Mackay (1990a) as well as Gantt and Nardi (1992) empirically investigated into the collaborative effects of tailoring, as sharing knowledge and artifacts. Our work is similarly structured but takes current developments into account. More recent work on collaborative tailoring and the related topic of software appropriation was especially done by Pipek and Kahler (2006). They did describe different shared scenarios (concerning usage, artifacts or infrastructure) that also lead to a need for awareness. While their work discusses this topic marginally, we wanted to focus more deeply on the role of awareness in collaborative EUD processes.

While existing research efforts investigated into organizations or closed groups as target for their research, we focus on groups, that act as social networks and are embedded in software ecosystems (Bosch 2009). Therefore we can address appropriation efforts in environments where large numbers plug-ins/addons from 3rd parties already exist. This results in a different view on collaborative tailoring efforts and awareness.

7.4. Conclusion

Our research shows that people may trust in particular recommendations of local peers, regarding modifications as plug-ins or addons. Both, Eclipse and WoW users facilitate third party tools, to orchestrate and maintain their sets of plug-ins or addons. These tools all share the same basic features. Curse for WoW, as well as the Eclipse Marketplace or Yoxos for Eclipse represent central repositories of components to modify the application. These tools keep track of what a user installs, they help keeping addons or plug-ins up to date and ensure an installation that is not broken after modifying. On the other hand, they miss to reflect the collaborative nature of modifying software that creates a demand for addon awareness and recommendations.

In small local or remote groups as the ones we examined, we found plenty of incidents were breakdowns, accidents or planned intervention functioned as a trigger that revealed a need for awareness. This resulted in discussions and recommendations. This was just "the tip of the iceberg", as we observed a constant and latent need for modification awareness in groups. And as there already exist tools to support some of the users EUD efforts, future research should suggest and evaluate possibilities to support *modification awareness*, as it could improve the reach of EUD.

III.

Designing Appropriation Support

8. Peerclipse: Tool Awareness in Local Communities[23]

Motivated by our research in the field of Eclipse users, we want to present our idea of Peerclipse – an Eclipse plug-in to support tool awareness and tool sharing in local communities, which are using Eclipse for software development.

23. This chapter has been published as: Draxler, Sebastian; Sander, Hendrik; Jain, Piyush; Jung, Adrian; Stevens, Gunnar; 2009. Peerclipse: Tool Awareness in Local Communities. In Supplementary Proceedings of the 11th European Conference on Computer Supported Cooperative Work. Vienna, Austria, pp. 19.

8.1. Grounded Design of Peerclipse

Eclipse is a good example for a highly flexible contemporary software system. It is based on an "everything is a plug-in"-philosophy and can be radically tailored by adding some of the thousands of additional components available on the Internet. Therefore, the user has the opportunity to adapt the software to the needs of his local working context. Unfortunately it is very difficult to keep track of the available components. In the CoEUD[24] research project we investigated how people use Eclipse as their daily working environment in Software companies. One of the key findings showed that, when the need for a new tool arose, suitable recommendations regarding tool selection, installation, and configuration were sought out from co-workers who also found themselves in similar working contexts. We especially could observe this in environments where software development was organized in agile teams. People did trust in their co-workers advice much more than in recommendations found on the Internet.

We therefore follow Mackay (1990a), Kahler (2001b) and our own observations, and suggest that tailoring support for tools like Eclipse should mirror the cooperative aspect of the users working environment.

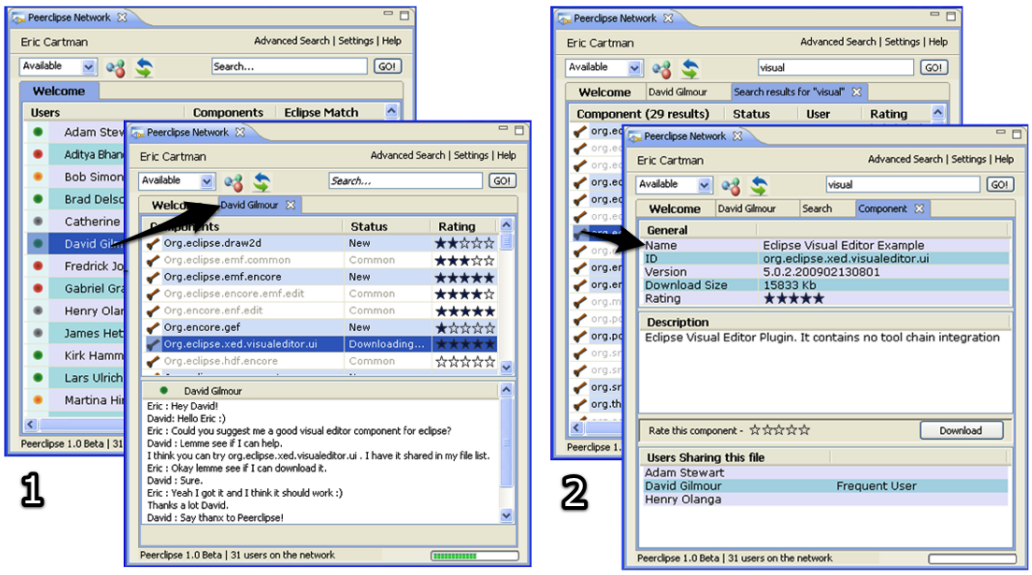

Figure 12: The Peerclipse User Interface.

Our empirical study has sensitized our design, demonstrating that tool awareness is an important issue. However there are often constraints that hinder an organic awareness, but should be supported by an appropriate technical infrastructure. Figure 12 presents

24. http://www.coeud.org/

first snapshots of the JXTA Peer-to-Peer based Peerclipse plug-in for Eclipse. It gives a good impression of how awareness support for tools used by co-workers could be integrated into Eclipse. It allows ad-hoc, Peer-to-Peer browsing, search and sharing of tools, used within the team. The team can be seen as a repository of tools in use and people with experience using these tools.

Scenario 1 (see Figure 12) shows Peerclipse, as the user Eric is searching for an appropriate user to look up for a desired component. In getting aware of his co-worker David, Eric starts to study David's configuration in detail. Scenario 2 (see Figure 12) illustrates a reverse activity of first searching the component and then looking up at the component's profile prior downloading it.

9. Supporting the Social Context of Technology Appropriation[25]

There is an increasing spread of flexible software applications that can be modified by adding components (sometimes called plug-ins or add-ons). A popular example in the software development domain is Eclipse, a flexible development environment that can be extended with literally thousands of different plug-ins. However, searching, installing and configuring new plug-ins requires complex overhead work that is only weakly addressed by existing support mechanisms. Recent research has highlighted the related practices of learning about new plug-ins and tailoring software tools as being highly cooperative, situated, socially embedded, and often connected to particular work situations. Based on an empirical study in small software enterprises, we develop an understanding of appropriation as a social and collaborative activity. We then suggest design principles for appropriation support that are grounded in the practices we have found in the field, and present a prototypical implementation of the concept that extends existing mechanisms of sharing tools and tool-knowledge.

25. This chapter has been published as: Draxler, Sebastian; Stevens, Gunnar; Stein, Martin; Boden, Alexander; Randall, David; 2012, Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems, Supporting the social context of technology appropriation: on a synthesis of sharing tools and tool knowledge. ACM, New York, pp. 2835-44. © 2012 ACM, Inc. http://doi.acm.org/10.1145/2207676.2208687

9.1. Introduction

Today, tailorability is an important part of software development. We have conducted a study on tailorability in the context of the Eclipse IDE, one of the most popular development environments that is widely used by software developers (EvansData 2007). Eclipse provides support for a broad variety of programming languages, connectors to other software engineering tools such as bug trackers, as well as project management functionalities (just to name a few). All these functionalities are provided by a huge number of available plug-ins, which can extend and adapt Eclipse to virtually any software engineering project (Murphy, Kersten & Findlater 2006). However, this flexibility also leads to problems for practitioners, who often need to deal with a considerable overhead in finding, selecting, installing and configuring the right plug-ins for a task at hand—always at risk of breaking their installations due to possible incompatibilities between plug-ins (Murphy-Hill & Murphy 2011).

As a consequence, discovering new features has become an inherent and non-trivial part of the daily work of software developers (EvansData 2007). This task can be time-consuming, confusing and involves a number of decisions concerning compatibility issues etc. Our data clearly shows that Eclipse users struggle with the necessary overhead work of managing their installations, keeping them up-to-date and reconfiguring them for new tasks. Hence, there is a strong case for understanding how people go about finding the plug-ins they need, what problems and constraints they have to deal with, and what kinds of support they rely on. In companies, appropriation is shaped to large extents by individual activities as well as by the formal structure and demands of the organization (Balka & Wagner 2006). Yet, in its very nature, appropriation is a highly social activity which is embedded in particular work-related situations and carried out mostly between peers or colleagues. Cooperative appropriation may be performed in numerous different and elegant ways. Examples are asking colleagues for help, purposefully discussing new tools, casually sharing tools over coffee breaks, using email or newsgroups to ask for help, or relying on Twitter for information about new tools (Draxler & Stevens 2011; Murphy, Kersten & Findlater 2006; Murphy-Hill & Murphy 2011).

As we will show in this paper, individual aspects of software appropriation like tailoring, configuration sharing or informal learning are covered well in the literature. However, the connections between these aspects are not well understood. At the same time, existing solutions for sharing and managing tools and configurations provide only fragmented support for the diverse facets of technology appropriation. Based on the example of Eclipse, we present a novel, lightweight approach that is based on a

holistic understanding of appropriation as an entangled, cooperative process of searching, becoming aware, installing, configuring, and learning how to use new tools. This approach is meant to overcome the fragmentations in the current design of appropriation support systems, filling a gap between solutions for general file sharing, specialized platforms for mass-distribution and configuration, as well as knowledge management approaches.

9.2. Related Work

Our work is grounded in the Marxian tradition of understanding appropriation as the social process of incorporating objects into one's life, including changes to the objects, caused by the changing modes of using it (Draxler & Stevens 2011; Stevens, Pipek & Wulf 2010). Configuring and sharing artifacts is seen as closely related to organizational learning in this tradition, and appropriation is understood as a holistic concept that includes adoption, tailoring, configuration sharing, tool awareness and informal learning. In this section, we want to give a brief survey of these inter-related aspects.

9.2.1. Sharing tools at work

In both CHI and CSCW, studying data sharing practices has a long tradition (Greif & Sarin 1987). Since peer-to-peer networks have become a mass-phenomenon, a special focus in HCI has been on music sharing (Brown, Sellen & Geelhoed 2001), addressing political and legal issues (like impacts on the music industry and digital right management), as well as hedonic issues (like socializing, intimacy, or expressing identity and personal experiences e.g. by creating mix-tapes) (Brown, Sellen & Geelhoed 2001; Voida et al. 2005).

From a technical viewpoint, Voida et al. (Voida et al. 2006) analyzed the different systems people use in practice for file sharing (like e-mail, shared folders, version control systems or peer-to-peer networks). By classifying these systems, they developed a scheme of categories (including scope, addressing, distribution, visibility and notification) that should be considered by the design of sharing tools in general. While such general principles are valuable, we will argue that the specific demands and pragmatics of sharing tools and customizations in the work context remain important. In the case of customizations the shared object is a hybrid one, since not only the files need to be shared (or copied), but also the contextualized information how to use the customized tool. Therefore, a technological solution to address this issue has to deal with knowledge sharing as well as pure file sharing.

With regard to customization sharing, several important studies from the 1990s showed that local social networks play an important role for adaptation and adoption of software in organizations (Gantt & Nardi 1992; Mackay 1990a; Wulf 1995). Within these networks, an informal division of labor exists where small groups of *lead users* create sophisticated customizations. These customizations are further adapted by *translators* to the needs of less experienced *users*, who usually do not adapt tools themselves, but profit from their colleagues' configuration work (Mackay 1990a).

As noted by Mackay, local work practices become objectified by the customizations and diffuse through the sharing practices: *"The exchange of customization files provides another example of how users in an organization share information about their preferred ways of interacting with software"* (Mackay 1990a). However, Pipek (2005) pointed out that the forms of usage and the use conventions are only partially manifested in the tailored artifact. In order to foster the collaborative appropriation and the negotiation of conventions, he argues for providing built-in mechanisms for recording and annotating usage, as well as use-discourse infrastructures to share these annotations with others.

In addition, as customizations affect the work within the group, Wulf (1995) argues that groupware systems should include a means for collaborative tailoring. However, because of the complexity and subtle interplay between individualization and group preferences, it remains an open research topic how general demands of collaborative tailoring can be realized adequately.

Kahler (Kahler 2001b) further elaborates Wulf's line of thought with regard to the special case of sharing individual customizations. He suggests technical as well as organizational means of support, including push and pull mechanisms (like sending tailored artifacts by email (MacLean et al. 1990) or by a built-in notification system as well as providing common repositories for configuration sharing (Kahler 2001a)), enabling the annotation of configurations and allowing to test the modifications in an exploration environment (Wulf 2000). Furthermore, he suggests raising the awareness of tailoring activities and fostering a tailoring culture by encouraging cooperation among colleagues as well as between users and local experts, and by recognizing tailoring efforts as important part of cooperative work with computers.

9.2.2. Finding new tools and learning how to use them

As indicated, appropriating new technologies refers not only to the physical installation of new tools but also to organizational 'learning', and specifically to transformations of work practices and the acquisition of new competencies (Dourish 2003). As

Lave and Wenger (Lave & Wenger 1991) have shown, such processes are often embedded in everyday work situations. Informal interaction within *communities of practice* turned out to be an important driver for knowledge exchange in this regard. The related learning processes have been outlined in an *apprenticeship* model that describes how apprentices learn from a master or colleague by peripherally participating in their practices.

Twidale has demonstrated how important these aspects are in the context of *"informal technical help giving between colleagues"* (Twidale 2005). Stressing the importance of the specific situation and the collaborative nature of learning activities, he showed how knowledge sharing could be understood flexibly and contingently. In contrast to the apprenticeship model that usually discerns fixed roles ("masters" and "apprentices"), Twidale discovered that actors tend to switch between these roles in practice. One example would be a student working in a software company. One the one hand, he could be seen as an apprentice learning about software development practices. On the other hand, in some situations he may also be seen as a master, introducing new technologies to his colleagues that he learned about in University. Such forms of learning therefore could often be better characterized as *peer learning*, instead of *apprentice learning*. Furthermore, Twidale observed that sitting together in front of a computer in order to work together and discuss work issues often determines a starting point and defines the context of the learning situation. Such forms of "over-the-shoulder-learning" (Twidale 2005) are important for the diffusion of work practices in an organization, as it enables people to discover and become aware of unknown tools or features in their social surroundings, and observe how to use them.

Similar issues have been studied by Murphy-Hill and Murphy (Murphy-Hill & Murphy 2011). They analyzed peer interaction in collocated and remote pair programming settings[26] according to related situated discovering and learning processes. Among other aspects, they identified sequences of *peer observation* and *peer recommendation*. In the first case, one programmer observes a colleague using a programming tool or feature that he/she was not familiar with. In the second case, the programmer is observed by a colleague and gets a suggestion with regard to a new tool or feature.

Such situations could be characterized as special cases of the over-the-shoulder-learning (Twidale 2005) that lead to the discovery of new tools and/or initiated learning processes about how to use these tools. In comparison to other information

26. Pair programming is an agile development technique where two software developers synchronously work together in front of one computer.

sources like internet forums or following Twitter, practitioners reported a preference for the local peer interaction because they valued the opinions and advice from colleagues' more. If possible, organizations should therefore support discussions about tools as a regular part of informal work communication as well as of regular team meetings. Furthermore, the results of Murphy-Hill and Murphy indicate a need for system design "*to make existing tools and environments more discoverable and distributed collaboration more effective.*" (Murphy-Hill & Murphy 2011).

9.2.3. Discussion

At the technical level, sharing tools and configurations is similar to file sharing in general (Voida et al. 2005). However, there are also important differences with regard to the related pragmatics. As tools have to be configured in order to match the individual needs or the IT infrastructure of the company, it is insufficient to share relevant files only. One also has to share the information how tools can and should be configured and used. As we have shown, such processes are often accompanied by forms of informal learning that are focused by concepts such as *legitimated peripheral participation* (Lave & Wenger 1991), *over-the-shoulder learning* (Twidale 2005) or *peer interaction* (Murphy-Hill & Murphy 2011). However, such concepts are hardly supported by general purpose sharing solutions like Napster or Dropbox that just support users specifying "*the what*, *with whom*, and *how* of sharing" (Voida et al. 2006, p. 2).

Even in the case of specialized platforms for software distribution (like Apple's App Store, the Firefox add-ons page or the Eclipse Marketplace), these aspects are only weakly supported. Rating and commenting functionalities are practically anonymous, making trustful interactions hard to establish. As a result, the features provided by these platforms do not have the same impact as recommendations from known persons or colleagues (Ozakca & Lim 2006; Rogers 2003). Also the existing configuration and installation managers that are integrated in some applications (like Firefox add-on manager or Eclipse update manager) primarily focus on the single user, and arguably neglect the sharing of tools and tool expertise within personal social networks.

In summary, we have identified a lack of support for the appropriation of new tools in situations where a large number of alternatives of varying use and relevance are possible. Current systems especially fail to address the social dimension of appropriation and fail to take into account the close interrelation between the physical sharing of tools, and the highly social and situated activity of sharing tool expertise. These shortcomings on the technical level also reflect shortcomings on the conceptual level. In particular, holistic concepts that provide theoretical accounts for the integration of configuration sharing and knowledge sharing features have been introduced just re-

cently covering only parts of the issue (Murphy-Hill & Murphy 2011). Hence, there is a need for further research that takes the complex interrelations between informal learning and collaborative tailoring into account. In the following sections, we present such an approach and demonstrate how it can be realized in the case of Eclipse.

9.3. Methodology

Our work adopts the combination of ethnographic studies of work practice and participatory design as outlined in the concept of Co-realization (Hartswood et al. 2008) and Design Case Studies (Rohde et al. 2009). As noted by Hartswood et al. (2008) knowing "what exists" can inform design, but cannot replace the practical experience of designing and using the actual artifact. In particular, the realization and appropriation of socially embedded software leaves ontic and epistemic traces that cannot be fully anticipated (Rohde et al. 2009). It has been suggested that design research comprises both retrospective and prospective elements (Balka & Wagner 2006). Prospectively, the purpose of theoretical and empirical knowledge is to sensitize and inspire design. Retrospectively, analytic work should make explicit *"what shows up […] as having been all along already implicit in that tradition [which is created by the ontic and epistemic traces of the realization process]"* (Brandom 2002). The purpose of retrospective analysis is further to clarify and elaborate the general lessons learned that are worth remembering. So prospection and retrospection follow a different logic, but taken together, they constitute the iterative design process (Rohde et al. 2009). Here, retrospectively, we discuss a design project, where we developed a plug-in solution for Eclipse that is able to support collaborative appropriation. The design realization followed an iterative, participatory approach that was mainly organized in three cycles:

9.3.1. Cycle 1: Concept Design

In the first cycle we studied the Eclipse appropriation practices in five organizations (3 small and medium enterprises, 1 large enterprise and 1 research institute). In 2 small and medium enterprises, we conducted participative observation for more than two weeks. We furthermore interviewed at least 3 developers in each organization. In addition, we conducted an online survey in which 138 persons participated. As part of this survey, 76 Eclipse installations were sent in to be analyzed. The findings of these studies are partially published in (Draxler & Stevens 2011; Stevens & Draxler 2010) (see also chapter 4). Their results show that appropriation is an important prerequisite for Eclipse users in order to get their work done, and that local social networks of colleagues are a hugely important source for support in this regard.

We then developed a first design concept enabling users to browse the Eclipse installations of their colleagues. We evaluated the concept at the research institute in a small team of 8 developers, who are located at the same floor. Following a Wizard of Oz approach, we manually collected information about all Eclipse installations in the team. Based on the installed plug-ins, we derived suggestions for interesting tools for the team members. The value of the suggestions was assessed by the participants, followed by interviews that aimed to identify possible collaborative appropriation features within Eclipse. We further developed a first demonstrator to illustrate the general idea.

9.3.2. Cycle 2: Interaction Design

In Cycle 2 we focused on improving the interaction with the demonstrator. Initially, we conducted a 3-hour Participatory Design workshop with 6 students who regularly used Eclipse. Additionally, we presented the existing demonstrator to 7 Eclipse users from 2 of the small and medium enterprises and the research institute. As a preliminary evaluation of the interaction concept, we presented screenshots of the user interface to the practitioners. Later, the participants explored the demonstrator by themselves. The user tests were followed by interviews to get an insight into user perceptions of the tool. These interviews followed the Laddering technique, took about one hour each, and included discussions about the general impression of the design, areas for improvement, motivational aspects and possible use scenarios. The usability of the new interface design was also evaluated by 5 of the previous participants in form of follow-up interviews. This part of the evaluation included open explorative tasks as well as 'solving' tasks such as plug-in searches, browsing user configurations and commenting on a plug-in. The specification of these tasks was informed by our previous empirical study and the participatory discussion about meaningful use scenarios.

9.3.3. Cycle 3: Field trials

In Cycle 3, we developed a fully functional prototype that is stable enough to be used in practice. This stable version is currently being evaluated at the research institute, where 8 users have integrated the plug-in into their daily working environments. The feedback of the users is being used to continuously improve the prototype.

9.4. Design Principles

This section describes the major design principles we have identified, based on the ongoing ethnographic and evaluative work conducted. As described above, this process

consisted of stages such as realizing and discussing prototypes and designing sketches with users, conducting field studies and surveys, as well as analyzing related work in the literature. As our understanding of appropriation practices reached a saturated level (Glaser & Strauss 1967), we elaborated a system of related issues that designers of supportive systems for collaborative appropriation might consider. In the following, we present the most essential parts of this design space analysis.

9.4.1. Facilitating collaborative appropriation

"[…] especially one person in the project was extremely experienced in using Eclipse. It would have helped me a lot to know which [tools] he was using. I just would have used the tool[s] he used and wouldn't have had to worry about it." (Participant of an interview study, beginning cycle 2, translated by authors)

In the software industry, technology often follows rapid innovation cycles that make it hard for practitioners to keep up to date in their relevant fields. Furthermore, tool competencies often vary among team members in software companies with regard to different degrees of expertise (see e.g. (Mackay 1990a)), but also with regard to the different specializations of team members (as we observed in our fieldwork). The dynamics and diversity of technology-related knowledge can also be seen in the working environment. For example, in our online survey the average Eclipse installation consisted of 418 plug-ins combined into 42 assembly units (so called Eclipse features), the average age of a plug-in version was not older than 6 months (Draxler & Stevens 2011; Stevens & Draxler 2010) (see also chapter 4). In addition, even among colleagues no installation was identical to the other.

In line with existing literature (Kahler 2001b; Mackay 1990a; Murphy-Hill & Murphy 2011; Twidale 2005), our fieldwork shows that tool adoption emerges mainly from ad-hoc circumstances in everyday work (Draxler & Stevens 2011; Stevens & Draxler 2010). Even when people were not actually working together on a common task, we observed developers sometimes 'having a look' at a colleague's machine to understand which tools he used for his daily work. If something attracted attention, communication was initiated about further explanations, help was taken and given, and customizations were shared with each other.

Support for collaborative appropriation should exploit this diversity, as it allows people to learn from each other. In particular, it should support the situated and social nature of appropriation in the context of modifying tools and learning new features in interaction with colleagues. In addition, design has to recognize the fact that appropriation is an important part of the 'work to make things work', but it is usually not the

primary work objective (Balka & Wagner 2006). Hence, the overall challenge is to facilitate situated appropriation, without distracting users from their primary work.

Based on our observations, supporting collaborative appropriation should consider the following design issues:

- **Workplace integration:** Supporting the interleaving of work and appropriation (Twidale 2005), a solution should be seamlessly and consistently integrated into the work context.
- **Sharing scope**: With regard to the scope of sharing, the most promising granularity is the organization, the group and the team that works closely together or has at least some personal contact.

With regard to tasks that should be supported, we identified three major areas:

- **Searching tool knowledge:** Features are needed for supporting users in finding the right information about the tool that might solve the problem at hand, and/ or persons who are willing and competent to give advice in the given case.
- **Appropriation awareness:** In order to facilitate serendipitous situations of discovering new artifacts by chance, opportunities for social interaction in the sense of a "tickets-to-talk" approach (Svensson & Sokoler 2008) should be provided.
- **Peer installation:** In order to support the exchange and tools between colleagues, a solution for installing tools that are already in use by peers is needed.

While some of these design issues may sound self-evident, they are disregarded by existing tools for appropriation support and customization sharing. For example, sending customizations via emails (as suggested by (MacLean et al. 1990)) is detached from ordinary work and involves the overhead of collecting the relevant files (or related links to repositories). Similarly, using shared file systems as local repositories (as suggested by (Kahler 2001a)) introduces extra burdens with regard to administrating and maintaining the repository. While the more recent integration of marketplace features in applications (like the Firefox built-in add-on manager or the currently released plug-in for accessing the Eclipse marketplace) is a step in the right direction for satisfying the needs of appropriation, these solutions do not support the sharing of tools and tool knowledge within the local context of a team or company, but rely on external repositories or error-prone and clumsy file copying.

9.4.2. Workplace integration

In order to reduce any extra efforts and preserve the work context, appropriation features should be closely integrated into the ordinary working environment and be easily

accessible in a casual manner. The integrated features especially allow the interleaving of work and learning from others by reducing the overhead for both the help-giver and the help receiver by 'looking over the *virtual* shoulder' (Twidale 2005). In order to be perceived as an integral component, the additional features should conform to the general *look & feel* of the working environment. Further a lightweight solution is needed that avoids additional burdens of configuring the system and/or maintaining extra information.

9.4.3. Sharing scope

With regard to the sharing scope that specifies what data should be shared with whom, we can distinguish three design alternatives in general: public, selective and subnet (Voida et al. 2006). Examples of public sharing solutions are websites, blogs or Twitter, where the content is published on the Internet for a public audience. Examples of selective systems are Emails, where providers intentionally select the persons the information should be shared with. An example of a subnet solution is the possibility to share pieces of music and whole playlists in the media player iTunes. It restricts the public audience to persons in the same network (typically colleagues or friends).

With regard to the pragmatics of collaborative appropriation, the drawback of the *selective* scope is that it is only appropriate for intentional interaction where the provider knows in advance which person the information is relevant to. This sharing mechanism facilitates serendipitous situations only weakly. In addition, the selective mechanism excludes newcomers who are not yet well connected to established local networks. The selective mechanism is therefore inappropriate for promoting the learning processes of newcomers within the community of practice (Lave & Wenger 1991).

In contrast, sharing information with the general public audience allows others (even strangers or people one is only weakly connected to) to become aware about one's activities. Hence, the public sharing mechanism enables a legitimate peripheral awareness (Lave & Wenger 1991) of appropriation activities. Yet, the public scope is too broad and lacks a common work context that is important for trust building and initiating peer-interaction (Draxler & Stevens 2011; Murphy-Hill & Murphy 2011). We therefore recommend providing features that make appropriation activities publicly visible within the scope of the organization and its work groups. Technically, this could be realized by a subnet solution. Although the benefits of subnet sharing are well known and used by some tools like iTunes, to our knowledge no tool sharing solu-

tion exists so far which has adopted this sharing mechanism with regard to software tools and configurations.

9.4.4. Searching tool knowledge

"I think this [used browsing tools] would be interesting for our company, since (…) we do work in quite different projects. On the other hand, we are talking about a small group in whose competences I trust—unlike a web portal such as the plug-in community, which contains all [Eclipse] plug-ins, but I don't know who rated or commented these. That is just useless." (Participant of an interview study, translated by authors)

In order to support the searching within the local context, we explored the concept of being able to browse the tools installed by colleagues. As illustrated by the transcripts above, all users stated that a browsing feature would be really helpful for them. In addition, the users recognized that this feature is not only relevant for appropriation, but also addresses a general problem in cooperative work: namely identifying hidden conflicts that are related to using different tools and configurations (Pipek 2005):

"I would have been interested in my colleague's [Eclipse] installation, since I need to be able to get and run the code that he produces. It would be useful to compare and match different Eclipse configurations […]" (Participant of an interview study, translated by authors)

As tool knowledge only partially manifests itself in the plug-ins a user has integrated so far, such features as 'browsing' and 'comparing tool configurations' should be supplemented by a means to document and discuss configuration settings, best practices etc. (Mackay 1990a). Knowledge additionally should be made persistent as the appropriation of new tools happens irregularly and often casually as a kind of invisible work. For example, our study revealed that people often could not remember when and how they did modifications to their Eclipse installations, even in case they viewed these modifications as highly important (and sometimes struggled to replicate them in case of re-installations or updates). Hence, we believe that there is a need for a local memory that makes tool-centric knowledge persistent and easily retrievable, and that provides information on who has created the information, and how he or she can be contacted.

In addition, the system should increase the visibility of others' tool expertise and provide cues for becoming aware of what kind of tools the colleagues are usually working with. The cues should make use of automatic recordable information like which tools a person has installed, and how long they have been installed. However, the automatic collected data alone does not necessarily qualify practitioners to offer help to their col-

leagues, so cues should also consider user-controlled information, e.g. in a way as outlined by one of our participants:

"... the system should somehow visualize that one has done a lot of annotations, as this implies and shows others that one is experienced." (Participant of an interview study, translated by authors)

9.4.5. Appropriation awareness

"Probably it would be useful to be able to send messages like 'I solved all our problems by…' if I discovered something new. […] The question would also be, if I get informed if someone else installed or likes something." (Participant of an interview study, cycle 2, translated by authors)

In the literature, awareness is mainly studied from the perspective of coordinating activities (e.g. Heath, Luff & Cambridge 1992; Olson & Olson 2000). In our case, we are more interested in a second, often neglected aspect of awareness about the work activities of colleagues, that of casual 'looking over the shoulder' (Murphy-Hill & Murphy 2011; Twidale 2005). Observing work practices and noticing individual tool modification activities in a team allows for legitimate peripheral participation and supports the learning and enculturation of newcomers (Lave & Wenger 1991).

Existing research results show that awareness support systems are especially useful in the context of distributed work, as the knowledge about each other's context decreases with distance (Ozakca & Lim 2006). However, the evaluation of our design concept in the first cycle shows that even small co- located teams would benefit from such support. As indicated in the transcript above, appropriation awareness is especially useful if the people work on different projects and profit from the each other's specific competences.

To reduce overhead, we argue that an information notification mechanism should be integrated, so that users do not have to fall back to communication channels like email *"to make sure that their indented recipient knows that a new file is available"* (Voida et al. 2006). Such a mechanism should support both sending explicit messages as well as providing notifications that are implicitly triggered by tailoring activities. The last issue is related to Wulf's concept of informing groupware users of someone else's tailoring activities by an integrated notification system (Wulf 1995). The major motivation is to support serendipitous situations of *peer observation* and *peer recommendation*. This means that the notification system should not just inform users about conflicting activities, but also support them in sharing information about newly discovered tools or solved problems within the work context.

Technically, awareness features could be realized as an awareness pipeline (Fuchs, Pankoke-Babatz & Prinz 1995) that grants users fine grained control of the implicitly triggered notifications as well as the filtering of received events. As users want to spend as little extra time as possible with appropriation work, the design of sophisticated but complex configuration options has to adhere to the principle of proportionality. Hence, the design of a lightweight solution should take into account that notifications about tailoring activities will only sparsely be triggered as users do not modify their working environments daily. In addition, design should take into account that the default appropriation activities should be visible in the subnet in order to foster collaborative appropriation (see above).

9.4.6. Peer installation

"If you start using software, it helps to see how colleagues are using it. It helps to get a first overview, and to configure the [Eclipse] environment in a way that enables you to work with it." (Participant of an interview study, translated by authors)

Studies conducted in the 1990's (Kahler 2001a; MacLean et al. 1990; Mackay 1990a) mainly focused on the sharing of self-created customizations. However, today it is easy to distribute customizations world wide. Instead of creating own solutions, users nowadays have the opportunity to look for existing solutions that relate to their problem and share the adopted solution with others.

Our fieldwork indicated that Eclipse users follow this trend, as they made intensive use of the growing market of freely available tools. The practitioners regularly modified their working enviroment by downloading new plug-ins and exchanged them with colleagues. However, the intention to exchange plug-ins was typically not planned in advance, but emerged in peer-interaction situations, where one observed an interesting feature in use by a colleague. Acquiring the relevant feature, however, requires knowledge of:

- Which plug-in implemented the feature in question?
- Does the plug-in in question depend on other plug-ins?
- What plug-ins need to be transfered, which have already been installed, in which version, and will this create conflicts?
- Were are the plug-in's files stored?

In principle, the Eclipse installations of peers contain all the requireded information and files to answer these questions and to (semi-)automize the task. However, we are not aware of any solutions that support the exchange of tools in the peer interaction

situations we observed in our study. Instead, users are forced to carry out clumsy and error-prone workarounds as outlined above.

Based on this observation, we conclude that there is a need for supplementing existing solutions with features supporting peer-installation in situations of peer observation as well as peer recommendation (Murphy-Hill & Murphy 2011). We therefore decided to provide peer-installation features that support users in installing plug-ins and features observed on someone else's installation, as well as in recommending useful features to other users.

9.5. The Collaborative Appropriation Prototype

In order to support workplace integration and facilitation of appropriate methods, described above, we adopted the general sharing experience idea of iTunes. Like the music sharing feature is directly integrated into iTunes, the suggested features should be integrated directly into Eclipse. In order to realize this seamlessly, the prototype was implemented as an Eclipse plug-in. It uses the native Eclipse user interface, and relies on typical Eclipse interaction concepts. The user interface consists of different so called views (e.g. to show other peers) and editors (e.g. to comment a plug-in), typical elements of the Eclipse interaction concept, which consistently integrate the appropriation features into the work context (see Figure 13 for an overview).

Also like iTunes, our prototype is realized as a lightweight peer-to-peer solution, where users do not need to configure anything apart from activating the sharing and setting a nickname (optionally users can provide a picture, full name as well as their location). Technically, the prototype uses a simple JXTA peer-to-peer architecture that supports subnet visibility (Gong 2001). The network connections between the users' installations are established automatically if the prototype is installed on their machines, as the peers broadcast their availability. The communication infrastructure allows users to browse each other's Eclipse installation within the subnet to foster the coincidental nature of just meeting someone in the office. In contrast to iTunes, our prototype optionally caches information about users, the tools they installed as well as their comments and ratings (see below). Thus, tool information can be browsed even when the people are currently offline.

9.5.1. Realizing search and annotation features

In order to support *looking over the virtual shoulder* (Twidale 2000) we implemented an Eclipse view that allows users to inspect what tools are used in the local context. Like a Skype contact list, this view consists of all (cached) users in the subnet includ-

ing name, picture and presence information (see Figure 13 on the left). By clicking on an entry, the view changes its appearance and shows the plug-ins used by the colleague (Figure 13 on the right gives the example of the plug-ins used by Daniel). Information in the list of tools includes the tool's icon as well as the name and parts of the description. From our usability study we know that this usually enough information for Eclipse users to decide if the tool is worth further exploration. If a user wants to have further information, he/she can click on an entry. This opens a detail view (see Figure 13 black rectangle on the right) that gives additional information including comments and ratings of other users (see below).

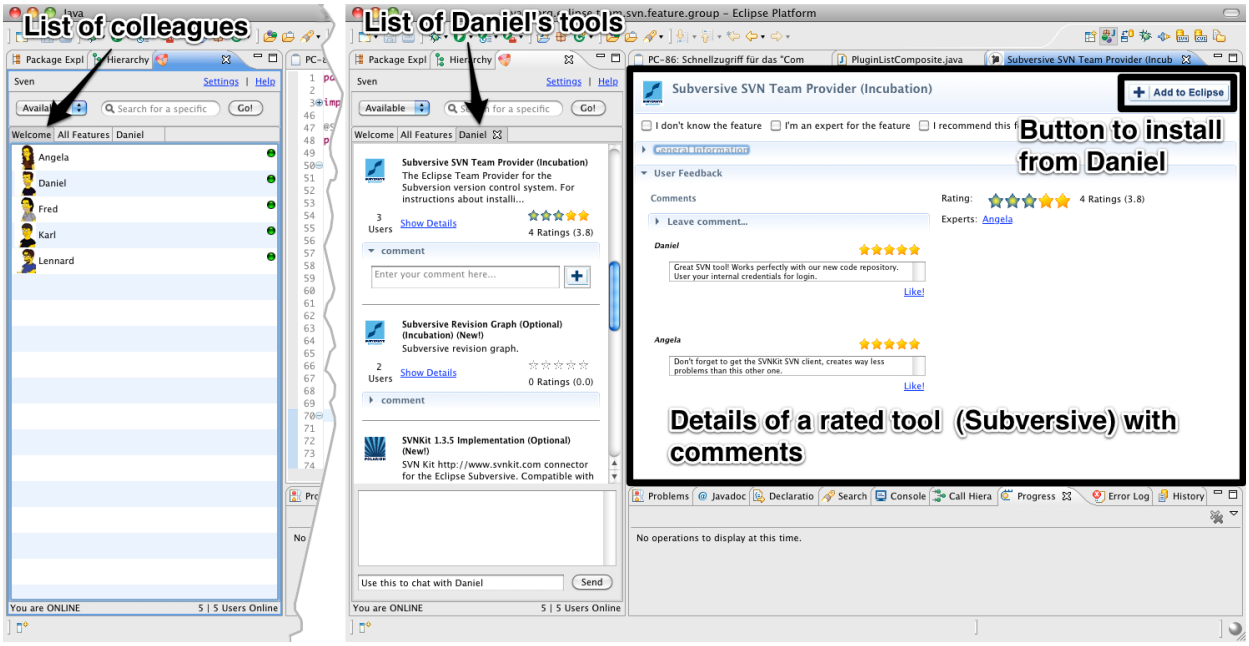

Figure 13: Integration of collaborative appropriation within the work context. Left: List of Eclipse users within the local network. Right: List of tools used by a colleague and a detail view showing the stored tool knowledge.

In general, these user interface elements enable users to explore the diverse Eclipse installations used in the subnet and look for tools that might be helpful for them. This is especially useful, as the experience in the group usually varies. To enhance the browsing experience, we further implemented different ways to filter and sort the results. For example, the user can search for keywords. This inverts the navigation, as the user can find a consultant by searching for a tool's name and therefore get help even without knowing in advance who is an expert on a particular topic. Further, it is possible to hide the tools that are used by both the user and his colleagues in the same version in order to enhance the awareness about differences in the configuration.

The list of tools can be sorted either alphabetically in order to simplify search, or by popularity in order to show potentially interesting tools. Currently, we measure popu-

larity by calculating the average rating given by users (see below) and by the number of colleagues who use the plug-in, if multiple plug-ins show the same average rating. In addition, it is possible to exclude tools from the result set that have not been rated or received bad ratings.

To capture and maintain a shared memory of the tool knowledge, we adopted the common concept (used for example by commercial marketplaces like Amazon) to allow customers to annotate and rate the products that are provided on the platform. We provided rating and commenting in the detail view (see Figure 13 right). During design workshops, we introduced several different ways of annotating, namely, comments, rating, tags, and categories. However, free comments and ratings were perceived as the most useful ways of sharing knowledge and experiences and have therefore been included in the prototype.

As illustrated by the quote below, it is important to understand the reasons for using such features as well as necessary configuration information:

"I would add tool related information [annotations]. For example information that is important for the initial configuration [of a tool]. Team members can benefit from this information as it will save time and money…" (Participant of an interview study, beginning of cycle 2, translated)

An important factor here is common work context. Colleagues, for instance, need to know what kind of pitfalls might occur when installing new plug-ins, or may find it useful to know that particular plug-ins are both useful and easy to install. Figure 13 (details box) presents an example of the different pragmatics. In this case one of participants used the implemented annotation feature to provide an instruction on how to configure the *SVN tool* in order to access the company's new source code revision control system.

Generally, restricting annotation features to the local organization has the effect that users are not forced to make knowledge explicit that could be taken for granted in the shared work context. This prevents the general problem of 'sticky information' (von Hippel 1994) and allows users to focus on more specific comments that are valuable for the local context. For example, formulations and instructions are highly beneficial for the local context, but do not make much sense without it. In their pragmatics, local annotations therefore differ from information that is targeting a public audience (like official help manuals or a users' review published on Amazon).

The shared work context also has an effect on the pragmatics of reading comments. In our fieldwork, for example, users often preferred to ask certain persons when seeking advice. Trust is often enabled by specific knowledge of the capabilities and characteris-

tics of colleagues. The system allows for such filtering and, if in doubt, supports to ask the author supplementary questions, to clarify some information or get additional advice.

9.5.2. Realizing awareness features

In order to raise awareness of the infrequent appropriation situations, the system sends notifications to local peers in cases of users tailoring their installations, or when they comment or rate tools (see Figure 14 for an example). By sending notifications to all colleagues, we provide a solution for the lack of peer interaction in non-collocated settings and increase their visibility at the team level of peer interaction, as the users do not need to be working in the same room or building to discover that a colleague is using a new or interesting tool.

Figure 14: Notification example of a tool (SWT Designer) that was commented and rated by the user Bart.

The notifications (we used Figure 14 as example) provide the user with different kinds of information:

- *Tool Awareness:* Users become aware of the tool itself (SWT Designer) and are able to look up more detailed information about it by clicking the corresponding link in the notification.
- *Peer information:* Users become aware of the fact that a colleague (Bart) has installed the tool. They can contact him or check his profile by clicking the corresponding link in the notification.
- *Additional context:* Users become aware that their colleague has provided comments to specify a certain work context, tasks, pitfalls or practices that have proven to be beneficial.

Notifications enable the user to navigate to relevant objects (a tool that might be interesting and a person who may be an experienced user) by clicking the corresponding link.

9.5.3. Realizing peer installation features

In order to support users to share tools, we implemented a peer install feature. It makes use of the circumstance that the users' installations hold all needed information and files for the exchange of plug-ins. Instead of forcing the user to switch to an external marketplace or to copy files manually, the user can install a tool from a colleague's installation by clicking a single button (see Figure 13 upper right corner). The functionality behind this button calculates the plug-in's dependencies, detects possible version conflicts, and identifies all the files that need to be transferred. As all needed files are stored on the colleague's machine, the prototype does not need an internet connection. Instead, all relevant files are copied using the peer-to-peer infrastructure and are afterwards installed on the local machine. To calculate dependencies between plug-ins, we used Eclipse's internal provisioning mechanism (P2), which is also used by the default mechanism to update and install plug-ins from market places. By reusing this functionality, the installation process using the prototype does not differ from the installation from any other source.

9.6. Evaluation

So far, we have conducted a limited evaluation in the research institute mentioned above. Even so, useful feedback has been obtained through workshops and usage tests during all phases of the design process. Users were generally positive and were particularly enthusiastic about the opportunity to browse each other's installations and be aware of each other's customization activities. The peer installation is more efficient and preferred over the clumsy and error-prone solution of copying files, and users do not feel distracted by additional appropriation features. In other words, the system acts to significantly reduce overheads in terms of time. It also has the valuable function of preventing installation crashes since one person's experiences can be easily propagated to others and necessary tools and plug-ins are provided. However, certain limitations are also becoming apparent.

One of the limitations that has become evident is the absence of a feature that can draw a connection between a user interface element and the underlying tool. For example a new Eclipse user needs help with a certain user interface element or functionality. However since Eclipse plug-ins are integrated so deeply it is not clear what tool/plug-in delivers this functionality and therefore he can't use the prototype to find an experienced user who uses the same tool. The same is true if a user wants to recommend certain functionalities but does not know which plug-ins contribute these. In a future version, we plan to adopt the approach of the Eclipse plug-in spy, a function-

ality which reveals the plug-in that is responsible for a certain user interface element by pressing a shortcut key.[27]

Another issue is the overly mechanistic implementation of the *subnet*, which needs to be more flexible than we have provided for. Changes in team membership; the involvement of external members (such as consultants to an organization) and casual sharing with outsiders (friends, university colleagues, and so on) are currently not well supported. In addition, the prototype does not yet support the sharing of configuration settings many Eclipse users have expressed a wish for.

9.7. Conclusion

Tailoring flexible software products is an increasingly important and common aspect of organizational work. As our study showed, Eclipse users have to deal with complex overhead work for maintaining and extending their working environments to the emerging needs of the daily work. This is a real problem that thousands of Eclipse users encounter regularly in their daily work. While over one thousand additional tools for Eclipse exist, it is hard to find relevant and useful ones. Furthermore, the users have to work out which conflicts may occur, what version of a tool is necessary, and what the tool can be used for.

In the case of Eclipse, the related sharing and learning practices turned out to be highly cooperative, situated, and grounded in the context of particular work situations. We have argued that because these issues are seldom addressed together, existing solutions only partially support the full context of appropriation. In other words, they largely fail to provide adequate appropriation support in organizations. Hence, there is strong evidence that better solutions for collaborative appropriation are needed and that we have to support all the related facets of finding, sharing and learning in order to support appropriation in practice.

Drawing on a detailed understanding of appropriation practices, we have presented design principles for supporting the social and collaborative aspects of appropriation. These principles were implemented in form of a prototype that addressed different aspects we have found to be important in the context of organizations:

- Appropriation awareness within the organization helps to systematically uncover customizations that are interesting for others.

27. Cf. http://www.eclipse.org/pde/incubator/spy/

- The active search for tools and tool expertise help to decide what to use and whom to ask, based on the people within the organization rather than relying on anonymous market places.
- Peer installation of customizations supports sharing customizations with colleagues in a safe way, without having to search for dependencies and without having to work with files directly.

Right now, the prototype is tested 'in the wild' in form of a long-term study. Early results are positive, though they are subject to the limitations we mentioned above. Right now it is too early for a summative evaluation of how the prototype is being used in practice. As many issues of appropriation support can only be studied in long-term field trials (due to the relatively rare and unconscious occurrence of appropriation), we plan to conduct further evaluations and iterate our design accordingly.

IV.

Conclusion

10. Conclusions

This chapter first presents the core findings of this work as a coherent whole and with regard to the main research questions of this dissertation. Following this summary, the results that contribute to a better understanding of appropriation, stemming from part II, are summarized and discussed in relation to the existing state-of-the-art of appropriation. The same is done with the findings that contribute to the design of appropriation support, stemming from part III, with regard to existing support approaches. The open questions and outlook subsection closes this work by discussing questions that could not be answered by this study or might have emerged as result of this work.

10.1. Research interests

Software ecosystems are a relatively new trend within software engineering. By introducing component-based systems in combination with commonly used platforms, software ecosystems allow software producers to join forces and form complex production networks. However, what does this mean from the users' perspective? Bosch (2009) openly suggests, that this change in production means will transfer large parts of the integration work to the users. In the past, it used to be the sole responsibility of the software manufacturers to integrate components into a fully working software system. With this shift, the significance of understanding how this change will transfer to the usage and appropriation of software increases. Will users have to run component-integration tests to ensure that two components work well together? Or will manufacturers provide different, easier mechanisms that help to find and select interesting components from repositories or markets and integrate them? What happens to practices like updating software components?

To date, the effects and consequences software ecosystems have on appropriation are poorly understood and not explicitly discussed within research on appropriation of software systems. Furthermore, it remains unknown in which ways supporting systems (e.g. the Eclipse update manager) help the users to cope with challenges such as integrating software components. Software Ecosystems with hundreds of components have already reached the mainstream of today's information technology usage. Browsers such as Mozilla Firefox or Google Chrome, games such as World of Warcraft and even phones (e.g. iOS, Android and Windows Phones) (see section 3.3) rely heavily on the addition of further components that modify functionality and user experience. Against this backdrop, and assuming that the end user should also benefit from software ecosystems, software engineers as well as human computer interaction experts need a better understanding of appropriation in software ecosystems and better design con-

cepts for supporting tools. Therefore this dissertation proposed the question: *how do users appropriate software systems that have been produced and developed by a software ecosystem?*

This was investigated through a field study in several organizations (see section 3.4). The micro-perspective of CSCW studies that results in a deep embeddedness of findings in context. The related difficulties of transferability has often been criticized. Due to this, but even more because of intrinsic motivation to compare the results with a different software ecosystem, this work proposed the question: '*how is appropriation constituted in similar ecosystems? / can we transfer these results to other ecosystems?'* Based on this, the results of the larger Eclipse study were complemented by and compared to a smaller study which was carried out in a different component-based software ecosystem.

As introduced, it is also unclear if the existing supporting tools are up to the task, which resulted in the question: *How should adequate supporting tools be designed to accommodate easy appropriation?* To answer this question, the field study was complemented by design workshops in order to reflect existing ideas and tools against the results. The results confirmed that existing support lacks important features, that are related especially to the group context (see chapter 9).

10.2. Appropriation revisited

10.2.1. Summary of Findings

One of the first findings was that nearly every Eclipse IDE user involved in one of the presented studies modified her/his Eclipse installation. Concerning the questionnaire study (see chapter 4), about 93% of all participants specified that they modified their Eclipse installations. During the extensive qualitative field study including interviews, observations and workplace inspections, not a single unmodified Eclipse installation was found.

The modifications were mainly the installation of additional components, so-called features and plug-ins, as well as the change of preference settings. Only one case of a plug-in, developed by a participant was found. Apart from this occurrence, all additional components originated outside the participant's personal or organizational networks. They were all created by an ecosystem of software manufacturers, connected by the common platform and partly by a common rhythm of development.

The participants' actions concerning appropriation were related to their personal preferences, their co-workers and their organizations. However, we found that they were also related to the software ecosystem:

1. which provided additional beneficial functionality through those plug-ins and
2. whose structures became recognized again, in case of plug-in related breakdowns.

We investigated the different levels that influence appropriation closer, as shown in the model depicted by Figure 15. The center is comprised of the personal social network or a work group; further outward, the department or even the whole organization (depending on size and contact between co-workers) can be found. The software ecosystem is situated in the outer layer, which also influenced appropriation by defining what is possible. Interestingly, we found that the influence on a user's appropriation seems not to decrease with the distance of a layer. However, the user's influence on the layer often decreased with the distance of the layer. E.g. the firewall rules at Gamma (see chapter 6) could not be changed, therefore some rules of the organizational layer could not be changed in this example.

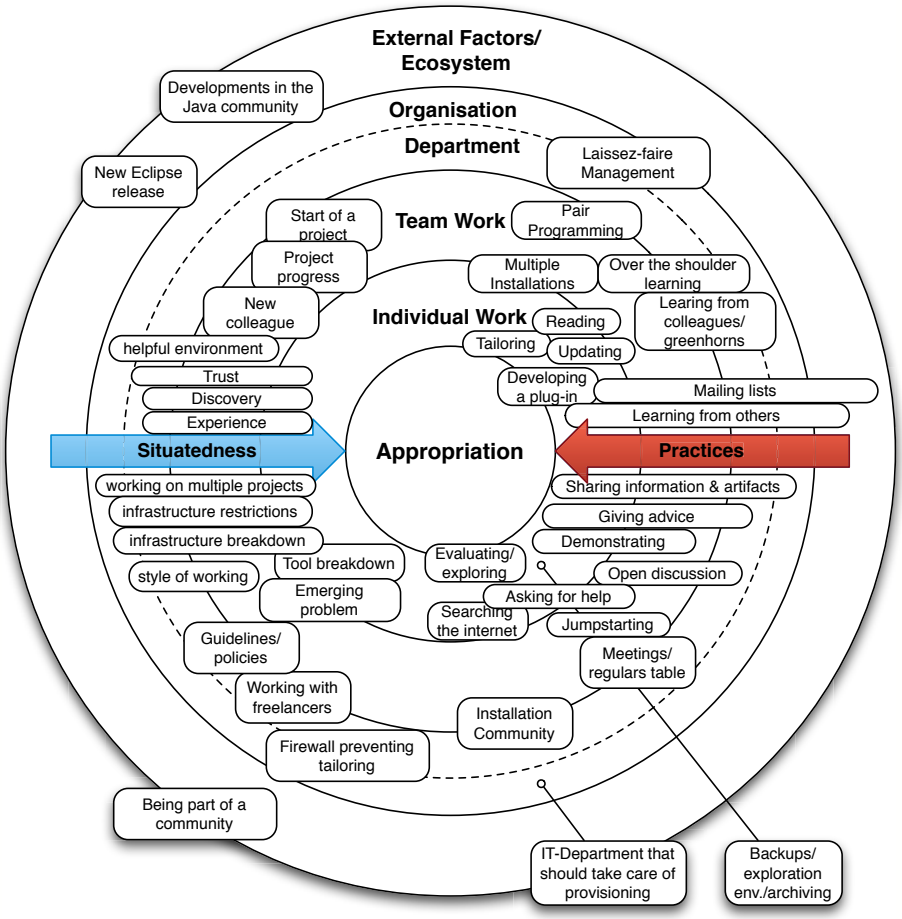

Figure 15: Layer model of situatedness and practices/strategies of appropriation of the Eclipse IDE, based on results of the field study.

During the field study, every layer presented an interesting frame that influenced the user's appropriation. Some users had strong preferences about their tools e.g. which plug-ins, frameworks technologies to use (which again sometimes required additional plug-ins e.g. for test coverage of unit tests). Sometimes, it happened that such personal preferences were incompatible and had to be omitted for the sake of collaboration with co-workers.

The group especially defined a frame that was related to the larger task to fulfill, e.g. a software development project. The choice of programming language for a project therefore usually defined how a certain Eclipse installation had to be modified in order to work within the group.

The already introduced inglorious firewall at Gamma, which prohibited most tailoring activities at the workplace, is an example of how the organization can set a frame of rules that influence appropriation (see chapter 6). An example that was more accepted within the group of participants was the provided source code management system, which also influences appropriation and which is usually defined on department or organization level.

The presented work tries to carve these layers out for better visibility (see chapter 4). What is especially interesting is the layer of a globally acting software ecosystem, consisting of an open platform and a network of manufacturers, equipped with decentralized component repositories and a distinct release rhythm.

This complexity of participating manufacturers, the changing API of the core platform and of the components[28], as well as the fast development rhythm was often seen to be boon and bane at the same time. On the one hand, the ecosystem stood for new and regularly updated and extended plug-ins that often grew very fast in terms of functionality. However, at the same time new features are accompanied by the need to carry out the necessary integration work. The Eclipse component model is designed so that, apart from installation, no integration work should be left after development and testing. However the complexity of the factors release rhythm and dependencies between components and their changing APIs created major breakdowns that had to be fixed by users so that they would work. The software ecosystem therefore created a tension amongst the study's participants, urging them to try new releases but at the

28. The Eclipse component model differs from others e.g. Firefox or Photoshop in a way that the components can depend on each other. An example is: component A provides a service that component B is supposed to use. Therefore, in some cases, component B needs component A to work properly.

same time urging them to protect the working tools as these represent a large part of their workplace.

Generated by this continuous struggle, but also by work tasks, stimuli from outside (e.g. Ecosystem) or personal interaction, various practices that contributed to appropriation could be observed. Interestingly, these practices were mostly embedded within the usual usage of Eclipse. Users learned from others through sharing a screen when coping with a complex task. Appropriation therefore could not be easily separated from usage. Chapter 6 draws a rich picture of the practices that were discovered during the field study.

These practices however could not solely describe appropriation within this context. Often it was the embeddedness of these practices in certain situations that occurred during work or organizational life which made appropriation accountable or reasonable. For example, a breakdown of the code repository at Alpha set the users in motion to get the company's infrastructure working again.

During the analysis of the gathered data, four categories had been created to classify the results. All categories contributed to describe certain actions or situational aspects that were part of daily work practices. *Learning and information gathering* focused on aspects such as personal help giving, regularly reading magazines and blogs or visiting conferences. *Tailoring* separated those actions that could be referred to as modification, configuration or tailoring in order to grasp the act of modifying that is often the observable part of appropriation. *Discovering* introduced serendipitous discoveries as a factor that influenced appropriation. Sometimes, pure luck led to learning something new and maybe changing the artifact in the process. *Collaboration* describes collaborative aspects that were part of all other categories (e.g. learning might have happened coincidentally when a problem was solved collaboratively; the Eclipse IDE was tailored due to the advice of a co-worker etc.).

Furthermore, a bird's eye view was taken during the analysis phase, that provided larger themes of appropriation efforts. The result is a classification model that defines different cultures of appropriation. The first class depicts very personalized approaches to appropriation, that only rely sparsely on collaboration. Secondly, a class of appropriation approaches that rely on formal collaboration by centralizing the appropriation management (partly), is described. Lastly, a class is given that builds upon collaboration within the group (e.g. exchange of interesting information) and trust (who disseminated this information?). Within this class, there is certain exchange concerning appropriation, but each person has to decide on his/her own what to make of it.

During this research effort, we had the chance to carry out a small but very similar study with one organization that used a completely different software system for comparison: a group of World of Warcraft players. Although we were searching for differences in their appropriation, we were mainly able to confirm what was found during the Eclipse IDE field study; rendering personal exchange, help giving and recommendations the most influential practices; trust and a collaborative task as very influential situational factors. Furthermore, both studies have shown that new components from the software ecosystems influenced the group's work/gameplay massively.

10.2.2. Discussion

By comparing the presented work to earlier studies, as carried out by Mackay (1990a), Gantt and Nardi (1992) or Kahler (2001a), it can be observed that earlier work mainly focused on the creation and sharing of self-created tailoring artifacts (such as configuration files or small components). Using Eclipse as the main example of this work, the corresponding artifacts would be plug-ins and preference settings. The results of those studies suggested that we would find several plug-ins that had been created by users. However, only a single occasion of a self-created plug-in could was discovered. Instead, the amount of readily available plug-ins from 3rd party manufacturers of the Eclipse ecosystem was huge[29]. In contrast to earlier studies, the appropriation of existing 3rd party plug-ins was the main aspect of appropriation to be found. As the ongoing End User Development research demonstrates (Lieberman et al. 2006; Pipek et al. 2009; Costabile et al. 2011; Dittrich et al. 2013), there is clearly a need for user created extensions in several areas of software usage. However, the appropriation of Eclipse revolves around the discovery and installation of existing components, as well as fixing of breakdowns created by those. That said, the appropriation of well-made, already existing components had not been investigated in such detail, before this study.

Furthermore, compared to earlier studies, everyone single person that was observed or interviews modified his Eclipse IDE installation. Compared with existing results (Mackay 1990a), this means an increase of tailoring activities. However, considering the domain that was investigated (software engineers, highly skilled in technical aspects, an urge to be on the bleeding edge of technology (Strübing 1992)), this was only to be expected. Remarkably, however, the results of the World of Warcraft study (see chapter 7) showed very similar results. For example, both studies showed a certain informal division of labor, as some Eclipse and World of Warcraft users were

29. We found about 1.000 Eclipse *features*, that act as containers for *plug-ins*, grouping these for easier installation. Each feature consists of multiple plug-ins.

more curious and eager to try out new things than others and shared their experiences. The general distinction between very explorative and more cautious users confirms earlier results (Mackay 1990a; Gantt & Nardi 1992; Rogers 2003), although was not not presented in detail as it was not the focus of the investigation.

The study has furthermore shown that software ecosystems are a new dimension that has to be considered when investigating appropriation and when designing appropriation support (Bourguin, Lewandowski & Lewkowicz 2013). The literature discussed earlier (see section 2.1) coined the term "software ecosystems", but mainly discussed this new trend from a manufacturer's point of view (Bosch 2009; Bosch & Bosch-Sijtsema 2010b; Bosch & Bosch-Sijtsema 2010a; Jansen, Brinkkemper & Finkelstein 2007; Jansen, Finkelstein & Brinkkemper 2009; Jansen, Cusumano & Brinkkemper 2013; Yang & Jiang 2007; Stein 2008; Boucharas, Jansen & Brinkkemper 2009). If we focus on studies that share a similar fundamental approach, stemming from the CSCW community, it can be noted that the early work only focused on such local contexts as the work group or the organization (Mackay 1990a; Gantt & Nardi 1992; Kahler 1995; MacLean et al. 1990). This is also due to the fact that global networking at that time was practically non-existent for end-users as well as manufacturers (Castells 2000, pp. 45-50). The existing results concerning sharing of knowledge and artifacts could be largely confirmed and extended by this work. Later work focusing on appropriation, especially (Pipek 2005), extended the focus of the topic and included networked communities, such as forums. While this added global networks as influential factors, the pressure, created by the Eclipse ecosystem, brings a new quality to influencing appropriation. Stevens (2009) is the first who actually referred to the trend of software ecosystems. However, the ecosystem is discussed as a black box that is part of the local context. The presented results add detail to Stevens' work: Artifacts that are part of appropriation efforts often come from outside the work group or the organization, but from within the global software ecosystem. As such, today's users are embedded into and influenced by such ecosystems, besides the group or the organization.

As predicted by Bosch and Bosch-Sijetsema (2010a), the introduction of software ecosystems shifted the responsibility for some integration work (e.g. work to make components operate together as a functional system) to the users. This study has shown in detail and based on empirical findings how this integration work is comprised for the Eclipse software ecosystem. It presented common problems (e.g. Eclipse not starting anymore after installing a plug-in) as well as strategies to cope with this (e.g. trying the respective plug-in within a blank, unmodified Eclipse installation and gradually modify). Integration work is a very knowledge-intensive task, as the ecosystem and the produced artifacts are in continuous motion. This confirms earlier results,

such as (Star & Ruhleder 1996; Balka & Wagner 2006). The focus on integration instead of creation of add-ons or tailoring artifacts (following e.g. the tradition of End User Development (Lieberman et al. 2006)) seems to demand a new connection between users and software engineers. Various researchers such as (Eriksson & Dittrich 2009; Stevens 2009; Dittrich 2014) argued for a more seamless integration of users and other stakeholders as consultants, software integrators and developers into the software development process. Some of these ideas include the software ecosystem level.

Considering the more detailed aspects of the results, the nature of tailoring activities and information exchange during the study is remarkable. Eclipse plug-ins were demonstrated to other users, discussed within team meetings or during coffee breaks. Overall, sharing expertise on how to install additional plug-ins and, even more importantly, which ones are the most interesting happened on a very informal basis. Similarities to the concept of informal learning are hard to ignore. The term informal learning is used to describe learning which takes place outside formal learning systems such as schools or universities, but are part of daily life (Garrick 2012). However, these informal learning activities benefit from users (during the study mostly junior developers), who had previous formal training in basic usage of the Eclipse IDE. The results of this study reinforce several other studies, in that they argue for the importance of informal learning and coordination at the workplace (Whittaker, Frohlich & Daly-Jones 1994; Hinn, Wang & Twidale 2004; Twidale 2005; Boden, Nett & Wulf 2007; Maalej & Happel 2008; Boden 2012; Bourguin, Lewandowski & Lewkowicz 2013). Similar to informal learning, the study helped us to understand practices that could be described as informal tailoring. Often a person's tailoring decisions affected others in some way. Despite this, many of these activities were not discussed with coworkers, but carried out on one's own initiative. Others becoming aware of this later generated a new chance to discuss the benefits of the tailoring decision and adopt the tailoring artifact.

Concerning the situatedness of appropriation, similarities to the concept of infrastructuring (Bowers 1994; Star & Ruhleder 1994; Karasti, Baker & Halkola 2006; Pipek & Wulf 2009) can be drawn. The concept of infrastructuring describes that technology, just like large infrastructures (e.g. the power network), disappears over time into the background and is not noticed anymore. However users can reflect their infrastructure in detail during a breakdown situation, for example. Pipek and Wulf (2009) described this boundary as the point of infrastructure. The study revealed that Eclipse users also reflected their infrastructure during breakdowns; especially which components or external hard- and software it comprised. This reflection helped to find solutions to cope with breakdowns during integration work. This work showed not only several

points of infrastructure, but also typical user strategies to cope with the underlying problem.

Another remarkable insight was that appropriation, even in work places, is not always need-driven, as implicated especially by research in the field of CASE tools (Kemerer 1992; Fowler et al. 2000; Robbins 2005, p. 121). Instead, various situations of opportunity-driven appropriation were observed. One example was switching to a new Eclipse IDE version just after it was released. Or the installation of a new plug-in intended to create a more comfortable situation for the user. Points of infrastructure that resulted from opportunities could often be traced back to a form of awareness of the work of others (see chapter 6). Eclipse users were often aware of their colleagues' actions or became aware of events within the Eclipse ecosystem, e.g. the annual release of the new Eclipse version every June.

In the past, many categorization schemes of tailoring or appropriation have been developed. Examples are (MacLean et al. 1990; Mørch 1997; Pipek & Kahler 2006). For this context, the model of Pipek and Kahler (2006) is most useful, as it focuses on the scope of tailoring activities as well as dependencies between actors. Tailoring is categorized into four scenarios: shared use, shared context, shared tool, shared infrastructure (Pipek & Kahler 2006). Each scenario describes a stronger need for collaboration in order for tailoring to be beneficial. When applied to the findings of this study, several of these scenarios can be applied at the simultaneously. Eclipse users form a community of interest as they are users of the same basic tool (even when it may be configured very differently). Furthermore, the shared context scenario applies, as the study's participants worked as groups (within their organizations) towards a common goal (to develop a piece of software). The shared tool scenario applies, as Eclipse use was intertwined with other tools as source code repositories, issue tracking systems shared by the whole team. Modification of these tools can be considered a shared tool scenario. At the same time, modifying these systems may create severe consequences for Eclipse usage. Due to this, it could also be classified as a shared infrastructure scenario. Therefore Eclipse appropriation can be considered as requiring careful collaboration efforts. However, the model does not include software ecosystems (or a similar force) as a surrounding factor and this would seem to create additional needs for collaboration as they add new levels of complexity and new urges to innovate which often have to be coordinated.

Within the same line of work (Pipek & Kahler 2004; Pipek & Kahler 2006), several suggestions to support collaborative tailoring have been given. While the authors deduced techniques for support (e.g. repositories of components or awareness mecha-

nisms), this work agrees with the results, but at the same time grounds the suggestions in an empirical qualitative field study.

10.3. Designing for collaborative appropriation

10.3.1. Summary of findings

Today's software provisioning approaches and standards, e.g. Cobit and ITIL (Bon 2004), can prevent users from appropriating software as part of certain measures pertaining to controlling IT within organizations. However, during the study, it was observable that users cared about their software and constantly tried to improve it. A tailoring culture (Robertson 1998) could be found in each of the participating organizations. The users viewed themselves as expert users of their tools, arguing that others would not be able to configure their tools the right way. For example, the developers at Gamma had a dispute with their IT department over the provided Eclipse versions, which were deemed too old. As a result many users were unhappy with other organizational units, as self-determined appropriation was hindered. While the goals of standardization wihtin workplaces are understandable, the study has shown that it may be of more use to support such tailoring or appropriation cultures in contexts of highly skilled workers who are able to self-organize themselves.

Furthermore, the study has shown that the integration of access to product communities into the product itself is a useful measure to reduce hurdles. It removes some of the usual media breakage that happens if a user has to switch from an application to e.g. a web browser in order to access the product community. Designing the Peerclipse prototype as an Eclipse plug-in brought some remedy to such typical problems. At the same time, because the prototype was integrated within the host software (in this case the Eclipse IDE), it was possible to capture a unique set of data (running plug-ins), in order to create awareness of what others use, which would otherwise not have been possible.

Other than typical recommender systems that present results based on a huge and mostly anonymous dataset (Linden, Smith & York 2003), Peerclipse reflects what other users within the near context modified. This unique function was created to take advantage of local trust and expertise relations between co-workers. Compared to global recommendations (e.g. a top 10 list of most used plug-ins on the Eclipse marketplace), creating awareness about the modifications made by co-workers allows Peerclipse users to understand and assess their co-workers' actions. As a result, the prototype can help to establish miniature product communities within organizations. This

way, interesting modifications at a certain person's workplace can diffuse more easily into the organization or work group.

The core mechanism that was chosen to deliver this functionality is awareness (Heath, Luff & Cambridge 1992). While the study has shown that appropriation was very important within this context, it has also shown that users discuss actions as tailoring efforts very unsystematically. The reasons for this may be that these actions happened very rarely and that plug-ins were often considered as still being under evaluation as modifications were explored for a while, before finally whether or not ir shoul dbe kept. However, creating awareness allowed the informed person to ask questions about the modification in order to learn more about it and to assess it for their own work context. As such, awareness of tailoring activities provided support in the sense of a *tickets to talk* (Svensson & Sokoler 2008) approach. It has to be noted that this notion differs from usual CSCW terminology, where "awareness" is usually used to describe a form of cheap coordination or *"effortless coordination"* of work activities (Gross 2013).

This approach showed interesting results when it was applied to a research department (Draxler, Sander & Stevens 2010), for instance. The members knew each other, but not all of them worked collaboratively. The tickets to talk approach of Peerclipse is based on data that was delivered by the host software. The Peerclipse prototype investigates its host Eclipse installation for changes since the last check and reports interesting changes to other users of Eclipse/Peerclipse. As such, Peerclipse raises awareness on newly installed artifacts. However, users draw connections between this information and the potential skills of the source of information (the person that just installed a certain plug-in). Therefore, people draw a connection between tools and their user's expertise.

10.3.2. Discussion

Peerclipse, the prototype developed as part of this work was designed to explore new supporting concepts for appropriation. As introduced in chapters 8 and 9, it mainly focuses on supporting work groups of collaborating Eclipse users. Examples are project teams collaboratively developing a software system. Peerclipse does not support the introduction of new information from the ecosystem into the group, but it does help to distribute innovations from ecosystem level that have found their way into the group, within the group. First of all, I would like to discuss why it was focused on the group, although most of this work clearly uncovered that the surrounding software ecosystem has a major influence on Eclipse users.

In the well-received study by Mackay (1990a), the exchange of experience and artifacts within the organization was the main category of results. This study has however

shown that in today's software ecosystems, exchange within the organization is used to transport new developments from the software ecosystem level into the group or to disseminate them within the group. This said, the exchange of appropriation experiences in personal networks, the work group or the organization are important and happen on a daily basis. However, as this research has shown, the surrounding ecosystem enhances this. It is a surrounding global networked community that introduces plenty of interesting additional opportunities for appropriation. The users can be described as being embedded within the ecosystem, just as they are embedded within their organization, work group or personal network. In the case of Eclipse, the software ecosystem was the origin of most innovations. However, the information of the availability of such innovations needed to disseminate within the group in order to be beneficial. Some innovations had been disruptive and made whole workplaces unusable. An exchange of experiences within the group was therefore considered very important. Hence no additional support to bring innovations from the ecosystem into the group was created. Instead, Peerclipse was designed to share innovations from the outside and related experiences with the group and to expose possible contact persons or experts, in case of problems occurring due to integration work.

Peerclipse was designed to use its host Eclipse installation as a repository for appropriation artifacts. Thus Peerclipse can extract and send installed, configured and running plug-ins to other Peerclipse users. At first sight, this is very similar to Kahler's (2001a) demand for repositories for tailoring artifacts that can be accessed by other users in order to choose beneficial modifications and apply them to the one's software. What Kahler (2001a) however did not discuss were certain aspects such as the creation and maintenance of such repositories. It is therefore unclear who should operate such repositories, how many there should be, etc. The creator of tailoring artifacts would be the first thought. However, this person reaps few benefits, yet has all the additional work to do, rendering success unlikely (Grudin 1988). Organizational units, such as the IT department, were mistrusted with the provisioning of Eclipse installations and therefore are also unlikely candidates in this context. Furthermore, most participating organizations were way too small to have a separate organizational unit for such tasks. Peerclipse copes with this situation by creating such repositories automatically, based on the existing installation. It therefore removes an additional step of work. Furthermore, Kahler (2001a) discussed repositories as being the main point for accessing artifacts, which is useful if these components are created within the organization. However, this study made clear that the integration work of already existing components emerged to be a very complex aspect of appropriation. The repository therefore also serves to mediate or share experiences with these components. This was implemented by incorporating discussion features directly into Peerclipse. These fea-

tures, combined with the peer-to-peer approach of Peerclipse, created a completely new design to support appropriation.

Recent work discussed the creation of appropriation infrastructures (Pipek 2005; Stevens 2009). Appropriation infrastructures can be seen as global spaces with the ability to host communities for the exchange of experiences and artifacts around (software) products. The design of Peerclipse took a different path, as the appropriation infrastructure concept as given does not explicitly include measures on beneficial awareness support. However, during the field study and corresponding design phases, we found that this was one of the most crucial aspects in supporting the investigated groups of Eclipse users. Peerclipse therefore attempts to support appropriation at group level and uses awareness mechanisms (balloon windows in case, anyone carried out a modification or commented on a plug-in) to provide a base for further face-to-face discussions among the group's members. This design was chosen as the study has shown that the global ecosystem does not substitute the kind of contextualized community provided by the local group.

The focus on providing awareness towards software-modifications within a work group of software developers is not completely new. The Palantír prototype (Sarma, Noroozi & Hoek 2003; Sarma, Bortis & Hoek 2007) already achieved this in 2003. It calculates a delta by comparing the software before and after modification, and uses this data to raise awareness on tailoring efforts. Peerclipse uses a similar mechanism; however it does not only exploit the existing modification data, it also empowers users to exchange experiences by attaching text-based discussions to these components. Peerclipse raises awareness on modifications, but also on subjective assessments of components, expressed as the users' experiences and ratings. Additionally, Peerclipse differs from Palantír, as it makes use of a peer-to-peer approach for exchanging the data that is used by the awareness mechanism. Therefore no centralized server is needed to run Peerclipse, which renders the setup and usage of Peerclipse very easy within a work group of small companies.

This underlying concept of appropriation awareness is furthermore an alternative concept to centralistic approaches as given in the Cobit or ITIL standards (Bon 2004). Centralized approaches for software provisioning within organizations define central actors (a person, a department, etc.) that are responsible for delivering the software to the users and also for choosing the software in question. However, this study has shown that within the field of Eclipse usage, the user is an expert of her/his software tools and can often assess best what is needed. Peerclipse therefore does not incorporate support for creating software profiles from which users can not deviate. Instead, awareness in combination with a user's ability to make sense of this information is the

driving mechanism behind Peerclipse, resulting in a self-organized software provisioning environment. Essential part of this self-organization were first of all personal help giving (c.f. Murphy-Hill & Murphy 2011) and learning through co-workers (c.f. Twidale 2005). However, such actions often relied on breakdowns and serendipitous situations as triggering factors. In addition to breakdowns and lucky accidents, the awareness feature is a third triggering factor, designed to create a more regular exchange of experiences, which could systemize self-organization.

A further reason for fostering the personal exchange of experiences within the group instead of within the global community of an appropriation infrastructure was the ability to exploit local context and trust relations. Both aspects enable people to assess the advice given by their co-workers (Bourguin, Lewandowski & Lewkowicz 2013; Mackay 1990b). As the awareness feature is able to report modifications within an Eclipse installation but also subjective comments that are attached to plug-ins, Peerclipse helps to find experts in case of problems. This approach borrows from modern knowledge management systems, like the expert finder (Reichling, Veith & Wulf 2007). However, this is the first work dedicated to support appropriation that makes use of this concept, only accompanied by ShareXP (Bourguin, Lewandowski & Lewkowicz 2013), which also heavily relies on the concept of sharing between co-workers, exploiting trust relations.

This work was shown throughout that existing support mechanisms in Eclipse, e.g. the update manager, can't keep pace with the development speed, complexity and integration work resulting from the software ecosystem. Peerclipse introduced new supporting features in order to support some aspects that are often neglected today. Examples are the focus on the group or sharing of already tailored artifacts, instead of using the manufacturers' plug-in repositories. Considering Peerclipse's unique features, it can be considered a reference architecture for lightweight, group-level appropriation infrastructures. As such, it features a unique set functionalities:

- A component sharing mechanism integrated within the application itself in order to reduce media breakage and speed up access time.
- Use of a peer-to-peer mechanism and the local network's boundaries to find co-workers without additional configuration efforts. Peerclipse users can therefore find each other and each other's modifications when working within the same boundaries.
- Existing modifications are shareable. The host software itself is therefore a self-contained repository for tailoring artifacts.

- Subjective ratings and comments for expressing and sharing experiences are attachable to the artifact in question.
- Contents are transferred through a peer-to-peer network to reduce the need of configuring Peerclipse (e.g. no account has to be created).
- Awareness features are implemented to spawn face-to-face discussions about appropriation in order to help each other to cope with integration work.

Examples for more globalized solutions to support Eclipse appropriation are Yoxos[30] and MyEclipse[31]. Both solutions have to balance very liberate groups in which each user can change her/his installation in a self-determined manner, as well as organizations that prefer centralized provisioning. Through the design of Peerclipse and close cooperation with the designers of Yoxos and informal exchange with the designers of MyEclispe (at the time called Pulse), sharing mechanisms of both systems have been refined.

10.4. Open Questions and Future Work

The presented work investigated the appropriation of the Eclipse IDE and software ecosystem from a user's perspective. Software ecosystems as well as other aspects that influence appropriation are still open research questions, despite the amount of existing work, as appropriation is highly embedded in the user's work context and social network.

Concerning the appropriation of Eclipse, the aspect of changing preference settings was excluded from this work. Through the lens of appropriation at the time, this was considered a too small modification and minor aspect of re-thinking/re-inventing the tool to focus this work on. However, the study has shown that preference settings are immensely important for the work of Eclipse users. Future work should apply a view that spans all different aspects of appropriation in order to understand the phenomenon and to design support. For example, preference settings could possibly benefit from the same awareness support as plug-ins. For Eclipse perspectives (sets of opened views/windows with their respective size and placing on the screen), their is even an existing prototype available that allows sharing of perspectives and related plug-ins and features (Bourguin, Lewandowski & Lewkowicz 2013).

30. https://yoxos.eclipsesource.com/discover.html (last accessed 2014/05/12).

31. https://www.genuitec.com/products/sdc/ (last accessed 2014/05/12).

Among other criteria, the Eclipse software ecosystem was chosen in order to move the attention towards a very active and complex software system. This allowed a thick description (Geertz 1973) of how users can act in such ecosystems. However, at the time Eclipse and the surrounding ecosystem comprised the technological pinnacle and, as such, comprised a tool for professionally trained software developers. Other ecosystems emerged in recent years which are fast moving into homes and work places. The amount of problems we found with Eclipse appropriation might represent what can happen to other software ecosystems. However, software ecosystems tailored more towards the end user might also create less problems. That said, examples as Firefox, iOS and Android are very active software ecosystems that demand to be studied in their own right. The same is true for different user groups, e.g. technically non-skilled users and different contexts e.g. software used within work or leisure contexts.

Additionally, not only should we investigate the currently existing ecosystems, but also compare the underlying component technology in order to understand what the right component model and composition model for end users might be. The current software ecosystems trend started mainly as a tool to give manufacturers more freedom, as 3rd party developers can be easily integrated. By comparing Eclipse and World of Warcraft to Firefox and Android, we can observe that Eclipse and World of Warcraft components can depend on each other. Firefox and Android do not use this mechanism. From a software engineering point of view, reuse is an important means to creating software that renders the second approach as the less optimal solution. From a user's point of view however, this simplified approach to components can considerably reduce complexity and might make the difference for appropriation and in the end for the success of such ecosystems. However, so far no formal investigations have been carried out to prove this.

The embeddedness of appropriation in context also demands the constant rethinking of the design of supporting approaches to accommodate appropriation. First approaches that also build upon appropriation awareness already exist. For example, AppBrain[32] investigates your smartphone and can tell your contacts what apps you have installed. However, the concept of miniature-communities that consist of team or work group members or small organizations have yet to be further explored. In line with Bourguin, Lewandowski & Lewkowicz (2013), I believe that these supporting concepts can be very successfully applied in different software ecosystems.

32. https://play.google.com/store/apps/details?id=com.appspot.swisscodemonkeys.apps (last accessed 2014/05/26)

Last but not least, the approach to share not single components, but sets of components that go well together or even complement each other is an interesting concept already being applied to the Firefox ecosystem (c.f. https://addons.mozilla.org). These sets of tailoring artifacts are assembled and shared by users and afterwards provided by the system for installation. We can't assess the success of these approaches without formal investigation; however, it seems to be a very promising idea for supporting larger communities of users who do not work collaboratively.

This work has shown that appropriation today can be very complex and confusing, even for technically skilled users. At the same time, the Eclipse IDE was the pinnacle of software ecosystems, employing a flexible component architecture, a very fast development rhythm and an open source model to include as many developers as possible. The results of this set of field studies may therefore be at the more extreme end of the scale, showcasing several breakdowns. As these events are observable, they are just the visible tip of the iceberg that is appropriation. Our understanding of the major, invisible, part of everyday technology appropriation is still very limited.

11. Bibliography

Aaen, I., Siltanen, A., Sørensen, C. & Tahvanainen, V.-P., 1992, Proceedings of the IFIP WG8.2 Working Conference on The Impact of Computer Supported Technologies in Information Systems Development, *A Tale of Two Countries: Case Experiences and Expectations.* North-Holland Publishing Co., pp. 61-93.

Andersen, R. & Mørch, A.I., 2009, Proceedings of the 2nd International Symposium on End-User Development, *Mutual Development: A Case Study in Customer-Initiated Software Product Development.* Springer-Verlag, Berlin, Heidelberg, pp. 31-49.

Anderson, C., 2009, *Free: The Future of a Radical Price,* First Edition ed. Hyperion, New York.

Asplund, F., Biehl, M., El-Khoury, J. & Törngren, M. 2011, Tool Integration beyond Wasserman, in C Salinesi & O Pastor (eds), *Advanced Information Systems Engineering Workshops,* Springer Berlin Heidelberg, pp. 270-81.

Balka, E. & Wagner, I., 2006, Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work, *Making things work: dimensions of configurability as appropriation work.* ACM, pp. 229-38.

Beck, K., 1999, *Extreme Programming Explained: Embrace Change,* US Ed ed. Addison-Wesley Professional.

Bennett, C., Myers, D., Storey, M.-, German, D.M., Ouellet, D., Salois, M. & Charland, P., 2008, A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams, *J. Softw. Maint. Evol.*, 20(4), pp. 291-315.

Bentley, R. & Dourish, P., 1995, Proceedings of the Fourth European Conference on Computer-Supported Cooperative Work ECSCW95, *Medium versus mechanism: Supporting collaboration through customisation.* pp. 133-48.

Bentley, R. et al., 1992, Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work, *Ethnographically-informed Systems Design for Air Traffic Control.* ACM, pp. 123-9.

Bergin, T.J., 1993, *Computer-aided Software Engineering: Issues and Trends for the 1990s and Beyond,* Idea Group Inc (IGI), Hershey.

Bertell, O., 1971, *Alienation: Marx's conception of man in capitalist society,* Cambridge University Press, London.

Boden, A., 2012, *Coordination and learning in global software development: articulation work in distributed cooperation of small companies,* Universität Siegen, Siegen.

Boden, A., Draxler, S. & Wulf, V., 2010, Multikonferenz Wirtschaftsinformatik 2010, *Aneignungspraktiken von Software-Entwicklern beim Offshoring - Fallstudie eines kleinen deutschen Softwareunternehmens.* Universitätsverlag Göttingen, Göttingen.

Boden, A., Nett, B. & Wulf, V., 2007, Second IEEE International Conference on Global Software Engineering, 2007. ICGSE 2007, *Coordination Practices in Distributed Software Development of Small Enterprises.* pp. 235-46.

Boehm, B., 2006, Proceedings of the 28th international conference on Software engineering, *A view of 20th and 21st century software engineering.* ACM, New York, pp. 12-29.

Bohnsack, R., 2003, *Rekonstruktive Sozialforschung: Einführung in qualitative Methoden,* utb, Stuttgart.

Bon, J.V., 2004, *IT service management: an introduction based on ITIL,* Van Haren Publishing.

Bosch, J., 2009, SPLC '09: Proceedings of the 13th International Software Product Line Conference, *From software product lines to software ecosystems.* Carnegie Mellon University, San Francisco.

Bosch, J., 2010, ECSA '10: Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, *Architecture challenges for software ecosystems.* ACM, Copenhagen, pp. 93-5.

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), pp. 67-76.

Bosch, J. & Bosch-Sijtsema, P.M. 2010b, Softwares Product Lines, Global Development and Ecosystems: Collaboration in Software Engineering, in I Mistrík, J Grundy, A Hoek & J Whitehead (eds), *Collaborative Software Engineering,* Springer Berlin Heidelberg, pp. 77-92.

Boucharas, V., Jansen, S. & Brinkkemper, S., 2009, IWOCE '09: Proceedings of the 1st International Workshop on Open Component Ecosystems, *Formalizing software ecosystem modeling.* ACM, New York, pp. 41-50.

Boudreau, M.-C. & Robey, D., 2005, Enacting integrated information technology: A human agency perspective, *Organization science*, 16(1), pp. 3-18.

Bourguin, G., Lewandowski, A. & Lewkowicz, M., 2013, Sharing Experience Around Component Compositions: Application to the Eclipse Ecosystem, *Int. J. Distrib. Syst. Technol.*, 4(4), pp. 15-28.

Bowers, J., 1994, CSCW '94: Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work, *The work to make a network work: studying CSCW in action.* ACM, New York, pp. 287-98.

Brandom, R., 2002, *Tales of the mighty dead : historical essays in the metaphysics of intentionality,* Harvard University Press, Cambridge.

Brooks, F.P. & Jr., 1987, No Silver Bullet - Essence and Accidents of Software Engineering, *Computer*, 20(4), pp. 10-9.

Brown, B., Sellen, A.J. & Geelhoed, E., 2001, Proceedings of the Seventh Conference on European Conference on Computer Supported Cooperative Work, *Music Sharing As a Computer Supported Collaborative Application.* Kluwer Academic Publishers, Norwell, pp. 179-98.

Bruckhaus, T., Madhavji, N.H., Janssen, I. & Henshaw, J., 1996, The Impact of Tools on Software Productivity, *IEEE Softw.*, 13(5), pp. 29-38.

Bueno, F., Cabeza, D., Carro, M., Hermenegildo, M., López, P. & Puebla, G., 1997, The ciao prolog system. reference manual, *The CLIP Group, School of Computer Science, Technical University of Madrid*, 1(7).

Burnett, M. 2009, What Is End-User Software Engineering and Why Does It Matter? in V Pipek, MB Rosson, B de Ruyter & V Wulf (eds), *End-User Development,* Springer Berlin, Heidelberg, pp. 15-28.

Burnett, M., Cook, C. & Rothermel, G., 2004, End-user Software Engineering, *Commun. ACM*, 47(9), pp. 53-8.

Buxton, B., 2007, *Sketching User Experiences: Getting the Design Right and the Right Design,* Morgan Kaufmann, San Francisco.

Büscher, M., Gill, S., Mogensen, P. & Shapiro, D., 2001, Landscapes of practice: Bricolage as a method for situated design, *Computer Supported Cooperative Work (CSCW)*, 10(1), pp. 1-28.

Carroll, J., 2004, Proceedings of the ECIS 2004, *Completing Design in Use: Closing the Appropriation Cycle.* pp. 337-47.

Carroll, J. et al., 2002, Proceedings of the 35th Annual Hawaii International Conference on System Sciences, 2002. HICSS, *Just what do the youth of today want? Technology appropriation by young people.* pp. 1777-85.

Castells, M., 2000, *The Rise of the Network Society,* Blackwell Publishers, Inc.

Chappell, D., 1996, *Understanding ActiveX and OLE: A Guide for Developers and Managers,* 1st ed. Microsoft Press, Redmond, Wash.

Chau, P.Y.K., 1996, An empirical investigation on factors affecting the acceptance of CASE by systems developers, *Information and Management*, 30(6), pp. 269-80.

Chervany, N.L. & Lending, D., 1998, CASE tools: understanding the reasons for non-use, *SIGCPR Comput. Pers.*, 19(2), pp. 13-26.

Chikofsky, E.J., 1989, *Computer-aided software engineering (CASE): software development,* The Institute of Electrical and Electronics Engineers, New York.

Clements, P.C. & Northrop, L.M., 2002, *Software Product Lines: Practices and Patterns,* Addison Wesley Professional, Boston.

Coleman, G. & O'Connor, R., 2008, Investigating software process in practice: A grounded theory perspective, *J. Syst. Softw.*, 81(5), pp. 772-84.

Costabile, M.F., Dittrich, Y., Fischer, G. & Piccinno, A. (eds.), 2011, *End-User Development - Third International Symposium, IS-EUD 2011, Torre Canne, Italy, June 7-10,,* Springer, .

Costabile, M.F., Fogli, D., Mussio, P. & Piccinno, A. 2006, End-user development: The software shaping workshop approach, in *End user development,* Springer, pp. 183-205.

Crowston, K., Wei, K., Howison, J. & Wiggins, A., 2008, Free/Libre open-source software development: What we know and what we do not know, *ACM Comput. Surv.*, 44(2), pp. 7:1–7:35.

Cummins, H. & Ward, T., 2013, *Enterprise OSGi in action : with examples using Apache Aries,* Manning, Shelter Island, NY.

Dahlbäck, N., Jönsson, A. & Ahrenberg, L., 1993, Wizard of Oz studies - why and how, *Knowledge-based systems*, 6(4), pp. 258-66.

Davis, F.D., 1989, Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology, *MIS Quarterly*, 13(3), pp. 319-40.

Dayley, L.D., 2012, *Photoshop CS6 bible,* Wiley Pub., Inc, Indianapolis, IN.

Des Rivières, J. & Wiegand, J., 2004, Eclipse: a platform for integrating development tools, *IBM Syst. J.*, 43(2), pp. 371-83.

Dittrich, Y., 2014, Software engineering beyond the project – Sustaining software ecosystems, *Information and Software Technology.*

Dittrich, Y., Burnett, M., Mørch, A.I. & Redmiles, D. (eds.), 2013, *End-user development 4th International Symposium, IS-EUD 2013, Copenhagen, Denmark, June 10-13, 2013. Proceedings,* Springer, .

Dittrich, Y., Lindeberg, O. & Lundberg, L. 2006, End-User Development as Adaptive Maintenance, in H Lieberman, F Paternò & V Wulf (eds), *End User Development,* Springer Netherlands, pp. 295-313.

Dittrich, Y., Vaucouleur, S. & Giff, S., 2009, ERP Customization as Software Engineering: Knowledge Sharing and Cooperation, *IEEE Software*, 26(6), pp. 41-7.

Dix, A., 2007, Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it - Volume 2, *Designing for appropriation.* British Computer Society, pp. 27-30.

Dourish, P., 1996, *Open Implementation and Flexibility in CSCW Toolkits,* Department of Computer Science of University College London, London.

Dourish, P., 2003, The Appropriation of Interactive Technologies: Some Lessons from Placeless Documents, *Computer Supported Cooperative Work (CSCW)*, 12(4), pp. 465-90.

Dörner, C., Heß, J. & Pipek, V., 2008, CHASE '08: Proceedings of the 2008 International Workshop on Cooperative and Human Aspects of Software Engineering, *Fostering user-developer collaboration with infrastructure probes.* ACM, New York, pp. 48-4.

Draxler, S. & Stevens, G., 2011, Supporting the Collaborative Appropriation of an Open Software Ecosystem, *Computer Supported Cooperative Work (CSCW)*, 20, pp. 403-48.

Draxler, S., Sander, H. & Stevens, G., 2010, Multikonferenz Wirtschaftsinformatik 2010, *Provisioning 2.0: Diffusion kleinteiliger Software in sozialen Netzwerken.* Universitätsverlag Göttingen, Göttingen, pp. 665-77.

Draxler, S. et al., 2009, Supplementary Proceedings of the 11th European Conference on Computer Supported Cooperative Work, *Peerclipse: Tool Awareness in Local Communities.* Vienna, Austria, pp. 19-20.

Draxler, S. et al., 2012, Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems, *Supporting the social context of technology appropriation: on a synthesis of sharing tools and tool knowledge.* ACM, New York, pp. 2835-44.

du Gay, P., Hall, S., Janes, L., Mackay, H. & Negus, K., 1997, *Doing Cultural Studies: The Story of the Sony Walkman,* SAGE publications India, New Delhi.

Eclipse Foundation, 2010, Eclipse Community Survey 2010, *http://www.eclipse.org/org/community_survey/Eclipse_Survey_2010_Report.pdf,* pp. 1-30.

Ehn, P., 1990, *Work-Oriented Design of Computer Artifacts,* L. Erlbaum Associates Inc., Hillsdale.

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

Eriksson, J., 2008, *Supporting the Cooperative Design Process of End-User Tailoring,* Blekinge Institute of Technology, Karlskrona.

Eriksson, J. & Dittrich, Y., 2007, Combining tailoring and evolutionary software development for rapidly changing business systems, *Journal of Organizational and End User Computing (JOEUC)*, 19(2), pp. 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: Applications for Organizational Progress*, pp. 19-34.

Erl, T., 2005, *Service-oriented architecture : concepts, technology, and design,* Prentice Hall Professional Technical Reference, Upper Saddle River, NJ.

EvansData, 2007, 3rd Annual Eclipse Global Enterprise Survey Research Findings Public Version, *http://wiki.eclipse.org/images/d/df/EDC_SurveyPreso_public_version.pdf,* pp. 1-17.

Eveland, J.D., Blanchard, A., Brown, W. & Mattocks, J., 1994, Proceedings of the 1994 ACM conference on Computer supported cooperative work, *The role of "help networks" in facilitating use of CSCW tools.* ACM, pp. 265-74.

Fernstrom, C., Narfelt, K.- & Ohlsson, L., 1992, Software factory principles, architecture, and experiments, *IEEE Software*, 9(2), pp. 36-44.

Fichman, R.G. & Carroll, W.E. 2000, The Diffusion and Assimilation of Information Technology Innovations, in RW Zmund (ed), *Framing the Domains of IT Management: Projecting the Future...Through the Past,* Pinnaflex Educational Resources Inc, .

Findlater, L., McGrenere, J. & Modjeska, D., 2008, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, *Evaluation of a Role-based Approach for Customizing a Complex Development Environment.* ACM, pp. 1267-70.

Finlay, P.N. & Mitchell, A.C., 1994, Perceptions of the Benefits from the Introduction of CASE: An Empirical Study, *MIS Quarterly*, 18(4), pp. 353-70.

Firesmith, D., 2005, Are Your Requirements Complete? *Journal of Object Technology*, 4(1), pp. 27-43.

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

Fischer, G., 2009, Proceedings of the 2nd International Symposium on End-User Development, *End-User Development and Meta-Design: Foundations for Cultures of Participation.* Springer, Heidelberg, pp. 3-14.

Fisher, A.S., 1991, *CASE: using software development tools,* 2nd ed. John Wiley & Sons, Inc., New York.

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

Forte, G. 1993, Integrated CASE Environments, in TJ Bergin (ed), *Computer-aided software engineering: Issues and trends for the 1990s and beyond,* Idea Group Inc (IGI), Hershey, pp. 3-52.

Fowler, L., Allen, M., Armarego, J. & Mackenzie, J., 2000, 9th Annual Teaching Learning Forum, *Learning styles and CASE tools in software engineering.* Curtin University of Technology, Perth, W.A.

Franke, N. & Piller, F., 2002, Configuration Toolkits for Mass Customization, *Setting a Research Agenda. Arbeitsberichte des Lehrstuhls für Allgemeine und Industrielle Betriebswirtschaftslehre, Technische Universität München*, 33, p. 4.

Friedrich, J. & Rödiger, K.-H. (eds.), 1991, *Computergestützte Gruppenarbeit (CSCW),* Vieweg+Teubner Verlag, Wiesbaden.

Frost, R., 2007, Jazz and the Eclipse Way of Collaboration, *IEEE Software*, 24(6), pp. 114-7.

Fuchs, L., Pankoke-Babatz, U. & Prinz, W., 1995, Proceedings of the Fourth Conference on European Conference on Computer-Supported Cooperative Work, *Supporting Cooperative Awareness with Local Event Mechanisms: The Groupdesk System.* Kluwer Academic Publishers, pp. 247-62.

Fuggetta, A., 1993, A Classification of CASE Technology, *Computer*, 26(12), pp. 25-38.

Gamma, E. & Beck, K., 2003, *Contributing to Eclipse: Principles, Patterns, and Plugins,* Addison Wesley Longman Publishing Co., Inc., Redwood City.

Gamma, E. & Beck, K., 2004, *Contributing to Eclipse: Principles, Patterns, and Plug-ins,* Addison-Wesley Professional, Redwood City.

Gantt, M. & Nardi, B.A., 1992, Proceedings of the SIGCHI conference on Human factors in computing systems, *Gardeners and gurus: patterns of cooperation among CAD users.* ACM, New York, pp. 107-17.

Garrick, J., 2012, *Informal Learning in the Workplace: Unmasking Human Resource Development,* Routledge, New York.

Geer, D., 2005, Eclipse becomes the dominant Java IDE, *Computer*, 38(7), pp. 16-8.

Geertz, C., 1973, *The interpretation of cultures: Selected essays,* Basic Books, New York.

Glaser, B. & Strauss, A., 1967, *The Discovery of Grounded Theory: Strategies for Qualitative Research,* Aldine Transaction, New Brunswick and London.

Glaser, B.G., 1978, *Theoretical sensitivity: Advances in the methodology of grounded theory,* Sociology press, Mill Valley, CA.

Glickstein, B., 2010, *Writing GNU Emacs Extensions: Editor Customizations and Creations with Lisp,* O'Reilly Media, Inc.

Gong, L., 2001, JXTA: a network programming environment, *IEEE Internet Computing*, 5(3), pp. 88-95.

Graham, W., 2012, *Beginning Facebook game apps development,* Apress, New York.

Greif, I. & Sarin, S., 1987, Data Sharing in Group Work, *ACM Trans. Inf. Syst.*, 5(2), pp. 187-211.

Grinter, R.E., Edwards, W.K., Newman, M.W. & Ducheneaut, N., 2005, Proceedings of the ninth conference on European Conference on Computer Supported Cooperative Work, *The work to make a home network work.* Springer-Verlag New York, Inc., Paris, France, pp. 469-88.

Grinter, R.E., Herbsleb, J.D. & Perry, D.E., 1999, GROUP '99: Proceedings of the international ACM SIGGROUP conference on Supporting group work, *The geography of coordination: dealing with distance in R&D work.* ACM Press, New York, pp. 306-15.

Gross, T., 2013, Supporting Effortless Coordination: 25 Years of Awareness Research, *Computer Supported Cooperative Work (CSCW)*, 22(4-6), pp. 425-74.

Grudin, J., 1988, Proceedings of the 1988 ACM Conference on Computer-supported Cooperative Work, *Why CSCW Applications Fail: Problems in the Design and Evaluationof Organizational Interfaces.* ACM, New York, pp. 85-93.

Hamilton, G., 1997, *JavaBeans API Specification 1.01,* Sun Microsystems.

Hansen, K.M., 2003, Proceedings of TIS 2003 Workshop on Tool Integration in System Development, *Activity-Centred Tool Integration – Using Type-Based Publish/Subscribe for Peer-to-Peer Tool Integration.* ACM, New York.

Hardgrave, B.C. & Johnson, R.A., 2003, Toward an information systems development acceptance model: the case of object-oriented systems development, *IEEE Transactions on Engineering Management*, 50(3), pp. 322-36.

Hartmann, A., Herrmann, T., Rohde, M. & Wulf, V. (eds.), 1994, *Menschengerechte Groupware - Software-ergonomische Gestaltung und partizipative Umsetzung,* Teubner, Stuttgart.

Hartswood, M. et al. 2008, Co-Realization: Toward a Principled Synthesis of Ethnomethodology and Participatory Design, in MS Ackerman, CA Halverson, MS Erickson & WA Kellog (eds), *Resources, Co-Evolution and Artifacts,* Springer London, pp. 59-94.

Heath, C., Luff, P. & Cambridge, G., 1992, Collaboration and Control: Crisis Management and Multimedia Technology in London Underground Line Control Rooms, *Computer Supported Cooperative Work*, 1, pp. 69-94.

Henderson, A. & Kyng, M. 1992, There's no place like home: continuing design in use, in J Greenbaum & M Kyng (eds), *Design at work: cooperative design of computer systems,* L. Erlbaum Associates Inc., pp. 219-40.

Henkel, J., 2004, Open Source Software from Commercial Firms – Tools, Complements, and Collective Invention, *Zeitschrift für Betriebswirtschaft*, 74(4), pp. 1-23.

Hinn, D.M., Wang, X.C. & Twidale, M.B., 2004, IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings, *Collaborative learning of computer applications in the contexts of work, learning, and play.* IEEE, pp. 201-5.

Houser, C. & Kalter, S.D., 1992, Eoops: an object-oriented programming system for Emacs-Lisp, *SIGPLAN Lisp Pointers*, V(3), pp. 25-33.

Iivari, J., 1996, Why are CASE tools not used? *Commun. ACM*, 39(10), pp. 94–103.

Jansen, S., Brinkkemper, S. & Finkelstein, A., 2007, IFIP Working Conference on Virtual Enterprises (Pro-VE): Establishing The Foundation Of Collaborative Networks, *Providing Transparency In The Business Of Software: A Modeling Technique For Software Supply Networks.* Springer, Boston, pp. 677-86.

Jansen, S., Brinkkemper, S. & Finkelstein, A., 2008, 15th Annual EuroMA Conference (Euroma 2008), *Component Assembly Mechanisms and Relationship Intimacy in a Software Supply Network.* European Operations Management Association, Groningen, Netherlands.

Jansen, S., Cusumano, M.A. & Brinkkemper, S., 2013, *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry,* Edward Elgar Publishing, Cheltenham.

Jansen, S., Finkelstein, A. & Brinkkemper, S., 2009, 31st International Conference on Software Engineering, *A Sense of Community: A Research Agenda for Software Ecosystems.* IEEE, Vancouver, Canada, pp. 187-90.

Jirotka, M., Gilbert, N. & Luff, P., 1992, On the social organisation of organisations, *Computer Supported Cooperative Work (CSCW)*, 1(1-2), pp. 95-118.

Johnson, R., 2003, *Expert one-on-one J2EE design and development,* Wrox, Indianapolis, IN.

Joshua, J.V., Alao, D.O., Okolie, S.O. & Awodele, O., 2013, Software Ecosystem: Features, Benefits and Challenges, *International Journal of Advanced Computer Science and Applications*, 4(8).

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

Kahler, H., 2001a, More Than WORDs – Collaborative Tailoring of a Word Processor, *Journal of Universal Computer Science*, 7(9), pp. 826-47.

Kahler, H., 2001b, *Supporting collaborative tailoring,* Department of Communication, Journalism and Computer Science, Roskilde.

Karasti, H., Baker, K.S. & Halkola, E., 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*, 15(4), pp. 321-58.

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

Kelle, U., 2001, Sociological Explanations between Micro and Macro and the Integration of Qualitative and Quantitative Methods, *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, 2(1).

Kelle, U., 2005, Emergence" vs. "Forcing" of Empirical Data? A Crucial Problem of "Grounded Theory" Reconsidered, *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, 6(2).

Kemerer, C.F., 1992, How the Learning Curve Affects CASE Tool Adoption, *IEEE Softw.*, 9(3), pp. 23-8.

Kernighan, B.W. & Plauger, P.J., 1976, Software tools, *SIGSOFT Softw. Eng. Notes*, 1(1), pp. 15-20.

Khalid, H. & Dix, A., 2010, 2010 International Conference on User Science and Engineering (i-USEr), *Learning from the appropriation of photologs.* pp. 311-8.

Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M.B., Rothermel, G., Shaw, M. & Wiedenbeck, S., 2011, The State of the Art in End-user Software Engineering, *ACM Comput. Surv.*, 43(3), pp. 21:1–21:44.

Kobsa, A. & Wahlster, W., 1989, *User models in dialog systems,* Springer-Verlag, Berlin ; New York.

Lave, J. & Wenger, E., 1991, *Situated learning: Legitimate peripheral participation,* Cambridge university press, Cambridge.

Le Berre, D. & Rapicault, P., 2009, IWOCE '09: Proceedings of the 1st International Workshop on Open Component Ecosystems, *Dependency management for the eclipse ecosystem: eclipse p2, metadata and resolution.* ACM, New York, pp. 21-30.

Lieberman, H., Paternò, F. & Wulf, V. (eds.), 2006, *End User Development,* Springer, Dordrecht, Netherlands.

Linden, G., Smith, B. & York, J., 2003, Amazon.com recommendations: item-to-item collaborative filtering, *IEEE Internet Computing*, 7(1), pp. 76-80.

Livingston, E., 1987, *Making sense of ethnomethodology,* Routledge & Kegan Paul, London.

Lundell, B. & Lings, B., 2004, Changing perceptions of CASE technology, *Journal of Systems and Software*, 72(2), pp. 271-80.

Maalej, W. & Happel, H.-J., 2008, A Lightweight Approach for Knowledge Sharing in Distributed Software Teams, *Practical Aspects of Knowledge Management*, pp. 14-25.

Maansaari, J. & Iivari, J., 1999, The evolution of CASE usage in Finland between 1993 and 1996, *Information and Management*, 36(1), pp. 37-53.

Mackay, W.E., 1990a, Proceedings of the 1990 ACM conference on Computer-supported cooperative work, *Patterns of sharing customizable software.* ACM, New York, pp. 209-21.

Mackay, W.E., 1990b, *Users And Customizable Software: A Co-Adaptive Phenomenon,* Massachusetts Institute of Technology.

MacLean, A., Carter, K., Lövstrand, L. & Moran, T., 1990, Proceedings of the 1990 SIGCHI Conference on Human Factors in Computing Systems, *User-tailorable systems: pressing the issues with buttons.* ACM, New York, pp. 175-82.

Malone, T.W., Lai, K.-Y. & Fry, C., 1995, Experiments with Oval: a radically tailorable tool for cooperative work, *ACM Trans. Inf. Syst.*, 13(2), pp. 177-205.

Markus, G., 1978, *Marxism and Anthropology: The Concept of Human Essence in the Philosophy of Marx,* Van Gorcum Ltd.

McClure, C.L., 1989, *Case Is Software Automation,* Prentice Hall, Upper Saddle River.

McIlroy, M.D., 1968, Mass produced software components, Garmich, Germany.

McKinsey Global Institute, Chui, M., Manyika, J., Bughin, J., Dobbs, R., Roxburgh, C., Sarrazin, H., Sands, G. & Westergren, M., 2012, *The social economy: Unlocking value and productivity through social technologies,* McKinsey Global Institute.

Melton, B., 2013, *Microsoft Office Professional 2013,* O'Reilly Media, Sebastopol, Calif.

Mendoza, L.E., Rojas, T. & Pérez, M.A., 2001, Organizational Indicators for CASE Tools Selection: A Case Study, *Revista Colombiana de Computación*, 2(2).

Messerschmitt, D.G. & Szyperski, C., 2003, *Software Ecosystem: Understanding an Indispensable Technology and Industry,* The MIT Press.

Messerschmitt, D.G. & Szyperski, C., 2005, *Software Ecosystem : Understanding an Indispensable Technology and Industry,* The MIT Press.

Muller, M.J., Haslwanter, J.H. & Dayton, T. 1997, Participatory Practices in the Software Lifecycle, in MG Helander, TK Landauer & PV Prabhu (eds), *Handbook of Human-Computer Interaction,* Elsevier, Amsterdam, pp. 256-97.

Murphy, G.C., Kersten, M. & Findlater, L., 2006, How Are Java Software Developers Using the Eclipse IDE? *IEEE Software*, 23(4), pp. 76-83.

Murphy-Hill, E. & Murphy, G.C., 2011, Proceedings of the ACM 2011 conference on Computer supported cooperative work, *Peer interaction effectively, yet infrequently, enables programmers to discover new tools.* ACM, New York, pp. 405-14.

Mørch, A., 1997, *Three levels of end-user tailoring: customization, integration, and extension, Computers and design in context,* MIT Press, Cambridge, MA.

Mørch, A.I., Stevens, G., Won, M., Klann, M., Dittrich, Y. & Wulf, V., 2004, Component-based technologies for end-user development, *Commun ACM*, 47(9), pp. 59-62.

Nardi, B.A., 2010, *My life as a night elf priest : an anthropological account of World of warcraft,* University of Michigan Press : University of Michigan Library, Ann Arbor.

NIST/ECMA TR/55, 1993, Reference Model for Frameworks of Software Engineering Environments, *http://www.cs.umd.edu/~mvz/pub/sp.500-211.pdf,* .

Northrop, L.M. & Clements, P.C. 2004, An Introduction to Software Product Lines, in RL Nord (ed), *Software Product Lines,* Springer Berlin Heidelberg, pp. 322-.

O'Brien, J.A. & Montazemi, A.R., 2003, *Management information systems: managing information technology in the business enterprise,* McGraw-Hill Ryerson, Limited.

Oevermann, U. & Allert, T., 1987, *Modern German sociology,* Columbia University Press, New York.

Olson, G.M. & Olson, J.S., 2000, Distance Matters, *Hum.-Comput. Interact.*, 15(2), pp. 139-78.

Oppermann, R., 1994, Adaptively supported adaptability, *International Journal of Human-Computer Studies*, 40(3), pp. 455-72.

Orlikowski, W.J., 1992, Proceedings of the 1992 ACM conference on Computer-supported cooperative work, *Learning from Notes: organizational issues in groupware implementation.* ACM, New York, pp. 362-9.

Orlikowski, W.J., 1993, CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development, *MIS Quarterly*, 17(3), pp. 309-40.

Orlikowski, W.J., 2000, Using technology and constituting structures: A practice lens for studying technology in organizations, *Organization science*, 11(4), pp. 404-28.

OSGi Alliance, 2005, *About the OSGi Service Platform,* OSGi Alliance, San Ramon.

Ozakca, M. & Lim, Y.-K., 2006, CHI '06 Extended Abstracts on Human Factors in Computing Systems, *A Study of Reviews and Ratings on the Internet.* ACM, New York, pp. 1181-6.

O'Mahony, S., Diaz, F. & Mamas, E., 2005, IBM and Eclipse (A), *Harvard Business School case*, pp. 906–1007.

Pilz, D., 2007, *Krisengeschöpfe: zur Theorie und Methodologie der objektiven Hermeneutik,* Deutscher Universitt s-Verlag GWV Fachverlage GmbH, Wiesbaden (GWV).

Pipek, V., 2005, *From tailoring to appropriation support: Negotiating groupware usage,* University of Oulu, Oulu.

Pipek, V. & Kahler, H., 2004, Tailoring together, *International Reports on Socio-Informatics*, 1(2), pp. 5 - 47.

Pipek, V. & Kahler, H. 2006, Supporting Collaborative Tailoring, in *End User Development,* Springer, Dordrecht.

Pipek, V. & Wulf, V., 2009, Infrastructuring: Towards an Integrated Perspetive on the Design and Use of Information Technology, *Journal of the Association of Information System (JAIS).*

Pipek, V., Rosson, M.B., Ruyter, B. & Wulf, V. (eds.), 2009, *End-User Development,* Springer Berlin, Heidelberg.

Poole, M.S. & DeSanctis, G., 1989, Emerging Technologies and Applications Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, 1989. Vol IV, *Use of group decision support systems as an appropriation process.* IEEE, pp. 149-157 vol.4.

Rai, A. & Patnayakuni, R., 1996, A structural model for CASE adoption behavior, *J. Manage. Inf. Syst.*, 13(2), pp. 205-34.

Ramírez Zúñiga, L., 2012, *Practice-centered support for indoor navigation : design of a ubicomp platform for firefighters,* Shaker, Aachen.

Randall, D., Harper, R. & Rouncefield, M., 2005, Proceedings of the Ethnographic Praxis in Commerce (EPIC) Conference, *Fieldwork and Ethnography in Design - A perspective from CSCW.* Redmond, USA.

Reckwitz, A., 2002, Toward a Theory of Social Practices: A Development in Culturalist Theorizing, *European Journal of Social Theory*, 5(2), pp. 243-63.

Reichling, T., Veith, M. & Wulf, V., 2007, Expert recommender: Designing for a network organization, *Computer Supported Cooperative Work (CSCW)*, 16(4-5), pp. 431-65.

Reichwald, R., Seifert, S., Walcher, D. & Piller, F., 2004, Proceedings of 2004 Hawaii International Conference on Computer Sciences (HICSS), *Customers as part of value webs: Towards a framework for webbed customer innovation tools.* IEEE, pp. 15-8.

Riemenschneider, C.K., Hardgrave, B.C. & Davis, F.D., 2002, Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models, *IEEE Trans. Softw. Eng.*, 28(12), pp. 1135-45.

Robbins, J. 2005, Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools, in J Feller, B Fitzgerald, S Hissam & K Lakhani (eds), *Perspectives on Free and Open Source Software,* The MIT Press, pp. 245-64.

Robertson, T., 1998, Shoppers and Tailors: Participative Practices in Small Australian Design Companies, *Computer Supported Cooperative Work (CSCW)*, 7(3-4), pp. 205-21.

Robinson, M., 1993, Proceedings of the Third Conference on European Conference on Computer-Supported Cooperative Work, *Design for unanticipated use.* Kluwer Academic Publishers, Norwell, MA, pp. 187-202.

Rogers, E.M., 2003, *Diffusion of Innovations, 5th Edition,* Original ed. Free Press.

Rohde, M., Stevens, G., Brödner, P. & Wulf, V., 2009, Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, *Towards a paradigmatic shift in IS: designing for social practice.* ACM, New York, pp. 15:1–15:11.

Röhr, W., 1979, *Aneignung und Persönlichkeit - Studie über die theoretisch-methodologische Bedeutung der marxistisch-leninistischen Aneignungsauffassung für die philosophische Persönlichkeitstheorie,* Akademie-Verlag, Berlin.

Ruel, H.J.M. 2002, The non-technical side of office technology: managing the clarity of the spirit and the appropriation of office technology, in CR Snodgrass & EJ Szewczak (eds), *Managing the human side of information technology: Challenges and solutions,* IGI Global, pp. 78-104.

Sarma, A., Bortis, G. & Hoek, A.V.D., 2007, Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, *Towards supporting awareness of indirect conflicts across software configuration management workspaces.* ACM, New York, pp. 94-103.

Sarma, A., Noroozi, Z. & Hoek, A.V.D., 2003, Proceedings of the 25th International Conference on Software Engineering, *Palantír: Raising Awareness among Configuration Management Workspaces.* IEEE Computer Society, Washington, DC, USA, pp. 444-54.

Schmidt, K. 2000, The critical role of workplace studies in CSCW, in P Luff, J Hindmarsch & C Heath (eds), *Workplace Studies: Recovering Work Practice and Informing System Design,* University Press, Cambridge, pp. 141-9.

Schmidt, K., 2011, *Cooperative work and coordinative practices,* Springer, Heidelberg.

Schuler, D. & Namioka, A., 1993, *Participatory Design Principles and Practices,* Lawrence Erlbaum Associates.

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

Sharma, S. & Rai, A., 2000, CASE deployment in IS organizations, *Commun. ACM*, 43(1), pp. 80-8.

Shaw, M. & Clements, P., 2006, The golden age of software architecture, *IEEE Software*, 23(2), pp. 31-9.

Shull, F., Singer, J. & Sjøberg, D.I.K., 2008, *Guide to advanced empirical software engineering,* Springer, London.

Sigfridsson, A., 2010, *The purposeful adaptation of practice: an empirical study of distributed software development,* University of Limerick, Department of Computer Science & Information Systems, Limerick.

Silverstone, R. & Haddon, L., 1996, Design and the domestication of ICTs: technical change and everyday life, *Communicating by Design: The Politics of Information and Communication Technologies*, pp. 44-74.

Sommerville, I., 2007, *Software Engineering,* Pearson Education.

Stallman, R.M., 1981, Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation, *EMACS the extensible, customizable self-documenting display editor.* ACM, New York, pp. 147-56.

Star, S.L. & Bowker, G.C. 2006, How to infrastructure, in *Handbook of new media: Social shaping and social consequences of ICTs,* SAGE Publications, New Delhi, pp. 230-45.

Star, S.L. & Ruhleder, K., 1994, Proceedings of the 1994 ACM conference on Computer supported cooperative work, *Steps towards an ecology of infrastructure: complex problems in design and access for large-scale collaborative systems.* ACM, New York, pp. 253-64.

Star, S.L. & Ruhleder, K., 1996, Steps Toward an Ecology of Infrastructure: Design and Access for Large Information Spaces, *Information Systems Research*, 7(1), pp. 111-34.

Star, S.L. & Ruhleder, K. 2001, Steps toward an Ecology of Infrastructure: Design and Access for Large Information Spaces, in J Yates & J van Maanen (eds), *Information Technology and Organizational Transformation: History, Rhetoric, and Practice,* SAGE Publications, Inc., Thousand Oaks, Calif., pp. 305-46.

Stein, A., 2008, Proceedings of the 2008 international conference on Digital government research, *Collaborative software ecosystems.* Digital Government Society of North America, Montreal, Canada, pp. 5-.

Stevens, G., 2009, *Understanding and Designing Appropriation Infrastructures: Artifacts as boundary objects in the continuous software development,* Universität Siegen, Siegen.

Stevens, G. & Draxler, S., 2010, Proceedings of the 9th International Conference on Designing Cooperative Systems, *Appropriation of the Eclipse Ecosystem: Local Integration of Global Network Production.* Springer, London, pp. 287-308.

Stevens, G. & Wiedenhöfer, T., 2006, Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles, *CHIC - a pluggable solution for community help in context.* ACM, Oslo, Norway, pp. 212-21.

Stevens, G., Pipek, V. & Wulf, V., 2009, Proceedings of the 2nd International Symposium on End-User Development, IS-EUD 2009, *Appropriation Infrastructure: Supporting the Design of Usages.* Springer, Siegen, pp. 50-69.

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

Stevens, G., Quaisser, G. & Klann, M. 2006, Breaking it up: An industrial case study of component-based tailorable software design, in H Liebermann, F Paternò & V Wulf (eds), *End user development,* Springer, pp. 269-94.

Stiemerling, O., 2000, *Component-based tailorability,* University of Bonn, Bonn.

Strauss, A., 1988, The articulation of project work: An organizational process, *Sociological Quarterly*, 29(2), pp. 163-78.

Strauss, A. & Corbin, J., 1996, *Grounded Theory: Grundlagen Qualitativer Sozialforschung,* Beltz, Psychologie-Verlag-Union, Weinheim.

Strauss, A.C. & Corbin, J.M., 1997, *Grounded Theory in Practice,* 1st ed. SAGE Publications, Inc, Newbury Park, CA.

Strübing, J., 1992, *Arbeitsstil und Habitus - zur Bedeutung kultureller Phänomene in der Programmierarbeit,* Wissenschaftliches Zentrum für Berufs- und Hochschulforschung der Gesamthochschule Kassel, Kassel.

Suchman, L., Blomberg, J., Orr, J.E. & Trigg, R., 1999, Reconstructing Technologies as Social Practice, *American Behavioral Scientist*, 43(3), pp. 392-408.

Svensson, M.S. & Sokoler, T., 2008, Proceedings of the 5th Nordic Conference on Human-computer Interaction: Building Bridges, *Ticket-to-talk-television: Designing for the Circumstantial Nature of Everyday Social Interaction.* ACM, pp. 334-43.

Szyperski, C., Gruntz, D. & Murer, S., 2002, *Component software: beyond object-oriented programming,* ACM Press ; Addison-Wesley.

Tan, L., Esfandiari, B. & Pagurek, B., 2001, WCOP 2001: Proceedings of the 6 th International Workshop on Component-Oriented Programming, *The SwapBox: A Test Container and a Framework for Hot-swappable JavaBeans.*

Taylor, F.W., 1911, *The principles of scientific management,* Harper & Brothers, New York.

Thompson, R., Higgins, C. & Howell, J., 1991, Personal Computing Toward a Conceptual Model of Utilization, *Management Information Systems Quarterly*, 15(1).

Trigg, R.H. & Bødker, S., 1994, Proceedings of the 1994 ACM conference on Computer supported cooperative work, *From implementation to design: tailoring and the emergence of systematization in CSCW.* ACM, pp. 45-54.

Trigg, R.H., Moran, T.P. & Halasz, F.G., 1987, INTERACT'87: Proceedings of the Second IFIP Conference on Human-Computer Interaction, *Adaptability and tailorability in NoteCards.* Elsevier Science Ltd, Amsterdam, p.723.

Twidale, M., 2005, Over the Shoulder Learning: Supporting Brief Informal Learning, *Computer Supported Cooperative Work (CSCW)*, 14(6), pp. 505-47.

Twidale, M.B., 2000, Proceedings of the Thirty-Third Annual Hawaii International Conference on System Sciences, *Interfaces for supporting over-the-shoulder learning.* Computer Society Press, pp. 33-7.

Vaughan-Nichols, S.J., 2003, The battle over the universal Java IDE, *Computer*, 36(4), pp. 21-3.

Venkatesh, V. & Davis, F.D., 2000, A theoretical extension of the technology acceptance model: four longitudinal field studies, *Management science*, 46(2), pp. 186-204.

Voida, A. et al., 2005, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, *Listening in: practices surrounding iTunes music sharing.* ACM, New York, pp. 191-200.

Voida, S. et al., 2006, Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, *Share and Share Alike: Exploring the User Interface Affordances of File Sharing.* ACM, New York, pp. 221-30.

von Hippel, E., 1986, Lead users: a source of novel product concepts, *Manage. Sci.*, 32(7), pp. 791-805.

von Hippel, E., 1994, "Sticky Information" and the Locus of Problem Solving: Implications for Innovation, *Management Science*, 40(4), pp. 439, 429.

von Hippel, E.V., 2005, *Democratizing Innovation,* The MIT Press, Cambridge, MA.

Weber, M., 1949, *The Methodology Of The Social Sciences,* Later Printing ed. Free Press, Glencoe.

Wenger, E., 1998, *Communities of Practice: Learning, Meaning, and Identity,* Cambridge University Press, Cambridge.

Whittaker, S., Frohlich, D. & Daly-Jones, O., 1994, Proceedings of the SIGCHI conference on Human factors in computing systems celebrating interdependence - CHI '94, *Informal workplace communication.* ACM, New York, pp. 131-7.

Wulf, V., 1994, Anpassbarkeit im Prozess Evolutionärer Systementwicklung, *GMD-Spiegel*, 24(3), pp. 41-6.

Wulf, V., 1995, Mechanisms for Conflict Management in Groupware, *Advances in Human Factors/ Ergonomics*, 20, pp. 379-84.

Wulf, V., 1999a, Proceedings of OZCHI´99, *On Search for Tailoring Functions: Empirical Findings and Design Concepts.* CSU-Publisher, pp. 105-11.

Wulf, V., 1999b, Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, *"Let's see your search-tool!" - collaborative use of tailored artifacts in groupware.* ACM, New york, pp. 50--59.

Wulf, V., 2000, Exploration environments: Supporting users to learn groupware functions, *Interacting with Computers*, 13(2), pp. 265-99.

Wulf, V. & Golombek, B., 2001, Direct activation: A concept to encourage tailoring activities, *Behaviour & Information Technology*, 20(4), pp. 249-63.

Wulf, V. & Rohde, M., 1995, Proceedings of the 1st Conference on Designing Interactive Systems: Processes, Practices, Methods & Techniques, *Towards an Integrated Organization and Technology Development.* ACM, New York, pp. 55-64.

Wulf, V., Pipek, V. & Won, M., 2008, Component-based tailorability: Enabling highly flexible software applications, *Int. J. Hum.-Comput. Stud.*, 66(1), pp. 1-22.

Wulf, V., Rohde, M., Pipek, V. & Stevens, G., 2011, Proceedings of the ACM 2011 conference on Computer supported cooperative work, *Engaging with practices: design case studies as a research framework in CSCW.* ACM, New York, pp. 505-12.

Yang, Z. & Jiang, M., 2007, Using Eclipse as a Tool-Integration Platform for Software Development, *IEEE Software*, 24(2), pp. 87-9.

Yin, R.K., 2009, *Case Study Research: Design and Methods,* SAGE.

Zelkowitz, M.V., 1996, Modeling Software Engineering Environment Capabilities, *Journal of Systems and Software*, 35(1), pp. 3-14.

Zettel, J., 2005, Methodology Support in CASE Tools and Its Impact on Individual Acceptance and Use: A Controlled Experiment, *Empirical Software Engineering*, 10(3), pp. 367-94.

Appendix I: Detailed overview of research sites

This Appendix contains a detailed overview of the different organizations that were part of the main empirical field study on Eclipse appropriation. It should adequately allow to assess the choice of research sites and help to understand some of the results. Within section 3.5, a much shorter overview is given that is sufficient to understand the results.

Alpha: Web application and groupware developer

Please note that Alpha is described in more detail within a journal publication (see section 4.6). I decided to place a shortened version of the description here, for easier comparison with the other organizations.

Alpha is a small software developing company, typical for the German software industry. The company employs about ten people. In addition, a small number of university students work part time for the company.

Alpha was founded twenty years ago as a spin-off to commercialize AlphaProduct, a web application that had been developed during a research project. The application was implemented using the Python programming language and is accompanied by several Java applets. AlphaProduct presents the main asset of the company. Alpha's marketing strategy is based on licensing AlphaProduct to customers and on creating customer-specific adaptations. Furthermore, Alpha is regularly involved in research projects during which AlphaProduct was continuously extended by innovative features.

Customer relations are mainly the duty of the CEO. He is supported by assistants who accomplish general administrative tasks and office work. The software development carried out at Alpha deals mainly with the maintenance and the continuous enhancement of AlphaProduct. This work is mainly conducted by eight software developers. The development work 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 (few days). Usually two or three developers were involved, depending on the complexity of the tasks. The second type covers projects to realize innovative features, typically conducted as a part of funded research projects. These projects are typically larger (in terms of the amount of work), run for a longer time and are carried out with other partners.

The CEO typically serves as the interface between the customer and the software developers in client projects. Usually, the client conveys his wishes and requirements to the CEO who discusses them with developers in order to make an offer. At the start of such a project, staffing depends on the actual workload of the developers as well as on their general expertise, prior knowledge and experience towards the task. To a large extend, it is the project staff who decide how the project is subdivided into individual work packages and how work tasks are assigned. The work is coordinated mainly by communication. In addition, other mechanisms are also applied, such as implicit coordination by using a shared repository. Wherever 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 by whichever person is available.

At the time of the study, parts of the company had started to use the Eclipse IDE for Java development work as well as for Python development. Nevertheless, several members of the core staff were still using text editors and did not plan to switch to the Eclipse IDE. We interviewed all Eclipse users at Alpha:

John is a computer science student. He works about 10 hours per week at Alpha. Within the company, he is employed in software development work. He is known as the Java and Eclipse expert within the company. He has worked with Eclipse for several years (since version 2.1) and perceives himself to be an experienced user.

Paul is also a student of computer science. He works about 2 days per week at Alpha. His tasks are software development using Python and Java. He described himself as a mature Eclipse user. Both John and Paul learned about Eclipse from programming courses at the university.

Peter is the **CEO** of Alpha and also one of the Eclipse users. He holds a degree in computer science. At the time of the study, he had been using Eclipse for 8 years and regularly participated in software development tasks at Alpha. Together with many of the older employees at Alpha, he took his first software development steps in console-based UNIX environments. While he turned towards Eclipse, some of his employees still preferred the old ways. He did not portray himself as an expert or power user when it came to Eclipse, but he appreciated it as a toolbox capable of integrating diverse tools, and offering a common look and feel over the whole range of tools.

Beta: Small web development company

Beta is a small software developing company. It was founded in 2002 by three students to realize a product idea they had worked on during a project at the university. This product idea won a national award. However, they unfortunately could not find a customer so that the idea never reached the state of a commercial product. In order to cope with this situation, they started working in different areas. In recent years, Beta has become a small but successful web development agency which provides services in three areas: E-Learning, Content Management and E-Commerce. Beta employs about six persons permanently, but had a quite high turnover rate in the past. In addition, the company cooperates with a network of freelancers, most of them are web and interface designers. The company has two executive directors with two areas of responsibility: one (Francois, not interviewed) takes care of the customer relations (more of a CEO role). The other one (Frank) is responsible for the technical issues (CTO).

Beta's main business strategy is to develop individual internet presences in cooperation with their clients. The result is uUsually implemented by means of an open source content management system and was hosted on Betas servers. In addition, training courses are provided which enable their clients to change content autonomously and make small adjustments to the web design themselves.

Typical content management projects for clients are carried out as follows: One of Betas directors (usually the CEO) meets with the customer and together they create a concept. Over time, this concept is further developed by Beta employees and also by integrating the input of freelance designers. In the end, the concept becomes a prose description of the project, enhanced by various user interface scribbles, as the company viewed their clients as very visual people. Such concepts include graphical design, interaction design, information about the target group and perhaps technical features. This concept usually presents the major contractual basis. One of the directors plans the division of labor and is responsible for overseeing the progress of the project. This involves negotiations between the client and the company's developers. For example if requirements are not clear enough or if other additional information is needed, the technicians and designers inform him of this either face to face in the office or via email, Skype or telephone. As the CEO serves as an interface to communicate with freelancers, he is included in the communication if the client has to be consulted in order to clarify the requirements. In general such projects have a duration of about two or three months.

In addition day-to-day business, Beta also implements and hosts a web-based business simulation game, used to to teach economics in schools. The server side of the game

was written in Java, while the web interface was realized in Flash. The initial development, the hosting and the sequential enhancement and adjusting of the game were paid for by a large enterprise as a part of their company initiative to sponsor education.

While the directors are responsible for the division of labor, there are certain other constraints. Various workers demonstrated special skills in specific areas of technology and were therefore engaged in certain projects. Beta made use of Eclipse, Java, PHP, Typo3 and other tools to deliver their typical projects.

Frank is one of the founders of Beta and fulfills the role of the chief technical officer (CTO). In cooperation with the CEO, he heads the company. While programming was of course not his main task at the time, he still sometimes did some work for clients himself and furthermore overlooked development work. He had been an Eclipse user since 2004. Whilst not considering himself to be an expert Eclipse user, he argued that he was experienced.

Daniel was a software engineering trainee at the time of the study. Before training at Beta, he had completed a three year training course at college to become an IT assistant. During his time at college, he learnt the basics of software engineering and programming. At the time of the study, he saw himself as an inexperienced Eclipse user; he had, however been using Eclipse for six months previous to the study. He initially worked on the modification of the content management system Typo 3 for two PHP-based client projects.

Gamma: Software development department at an insurance company

Gamma is an insurance company employing around 1,200 people. Although their main business is not software development, they still had a need for specialized software applications e.g. for insurance calculations, to connect brokers and end customers or to support customer relationship management. The software applications necessary to achieve this were developed and maintained in-house by a 90 person software development department. Within the department, the development and maintenance work was organized in different development teams. Contact between development teams is not systematized whereby some development teams are in contact with each other while others are not. There are no comprehensive formal rules that apply to all project teams or their mode of working. During the last two years, Gamma has developed a common service layer, used by some development teams to gain access to certain aspects of the business data.

We interviewed developers of one team that was working on two different projects – a business-to-business portal for brokers as well as a portal for end customers. The ongoing development started more than a year ago for both projects. Both projects are based on Java enterprise technology and the developers used Eclipse as their main development tool. In the beginning there were about ten people involved: a project leader, two designers and seven software engineers. Several of these people did not work for the insurance company itself, but for the consulting company Delta (see next section). When these projects started, some of Delta's developers joined Gamma's development team for the purpose of technology consulting, combined with practical hands-on training and knowledge exchange while working. The crucial phase of both projects already lies in the past and most of the external developers (from Delta) as well as the designers have left the project team. One person, employed by Delta, is still supporting the project team: the project leader. His job on the hand is to steer the project of course, but at the same time he introduced some changes in the use of tools as well as development methods. He introduced the agile development method Scrum and unit testing, as well as certain new frameworks and tools that were supposed to make development work easier. In the beginning, heading the project was a full time job but at the time of the study he was supporting the development, improvement and maintenance of the system, like the other software developers at Gamma. Both the project leader as well as the people interviewed were all trained software engineers, each holding a university degree in computer science.

Jan is the project leader for a development project and employed by Delta. While being paid by Delta, he is actually working full time at Gamma's site. Since 1998 he has been developing software, especially using the Java programming language. In 2001 he started working for Delta, executing consulting work and software development at the customers' sites.

Sven is a software developer and so-called architect. Next to his development duties, once a month he tailors a new Eclipse IDE exactly to the needs of the project team. This includes downloading a new Eclipse IDE release (at home), installation of additional plug-ins, configuration of preference settings, archiving the whole installation again, carrying the archive into the company on a flash drive and copying the new package to the company network storage.

Dominik is a software developer and so-called architect. When it comes to tailoring Eclipse, he relies completely on Sven, whom he calls the team's *Eclipse configuration representative.*

Delta: Software development and consultancy company

Delta is a medium-sized company, employing approximately 50 people including students and external workers. It was founded in 1999 and is closely related to a university, to the point that some rooms of the university are even used as Delta's offices. One of its founders is a professor at the university and the company itself is very engaged in different research projects.

Their main business is customer-specific software development, technology and development-related consulting as well as coaching and training courses for software development. In the past, Delta was engaged in developing software used in banking, insurance (see Gamma) and public services as well as the development of control rooms to monitor high-tech machinery (see Epsilon). Software development is usually organized in different development teams. Development work is hierarchically organized. It is generally the CEO who contacts the customer and is the person in charge whenever anything goes wrong. Project leaders work very closely with the customers, often to the point of working full time at the customer's site. The same is true for many software developers.

The CEO's idea of technology and development training encompasses members of the company working at the customer's site for quite some time if necessary, in order to teach certain technological (e.g. Eclipse platform development) and methodological know-how (e.g. the agile development approach) to the customer's employees. For example, the project leader interviewed at Gamma was actually one of Delta's employees.

Another of Delta's specialities is the development of software built on top of Eclipse technology. While not every person at Delta was involved in this line of work, each person we interviewed and observed had previously developed new plug-ins for the Eclipse platform. This renders Delta's employees a very experienced group of Eclipse users in the study. During this study, we followed one of Delta's project teams. In collaboration with their customer (Epsilon), they were developing a large and very flexible software application for a control room, based on Eclipse. As the machinery controlled by the operators changed constantly, the aim was to give the control room operators a very flexible tool. At the time, the Eclipse platform was a good base to deliver just that. In order to achieve this goal, Eclipse was used as a base for the application. New plug-ins delivered functionalities. A total of six people were working in this project team in cooperation with several other people at Epsilon. Everyone involved in the software development has direct contact with the customers. One of

Delta's specialities was to engage in pair programming with mixed teams (Delta and customer).

Delta did not establish company-wide regulations or formal rules concerning tool usage and configuration. The development teams on the other hand use informal agreements.

Martin is a software developer and was 23 years old at the time of the study. He is involved in training and coaching activities. He has used Eclipse since studying at university (for about 5 years).

Chris is a student of business informatics, working at Delta. He had worked in software development for several different companies for about seven years previously. He had used Eclipse ever since being involved in software development.

Karl holds a degree in computer science and works full time for Delta. He is involved in several projects, but mainly works for a larger development project at Epsilon, in collaboration with Chris. He is very active in exploring new technologies and has been an Eclipse user for several years.

Epsilon: Machinery maintenance department at large research site

Epsilon is a large research institution which mainly investigates in particle physics. At the time of the study, Epsilon employed about 1,700 people. Additionally, about 3,000 external workers visit Epsilon for purposes of work or research throughout the year. We focused on a small group within the organization that is responsible for parts of the cooling infrastructure. This cooling system is a significant component as the particle accelerator could not be run without it. The cooling system is a huge machine (spanning the whole accelerator ring) that cools down the various components of a particle accelerator. The group started several years ago with three people. At the time of the study, 20 people were employed to develop the hard- and software necessary for operating and monitoring the cooling machinery. The group was divided into operators and developers. During the study, we interviewed the development team.

The group consists of people from diverse backgrounds and with a range of skills. Some come from a physics background, some from (mechanical or electrical) engineering, but most of them hold a software engineering or computer science degree. They did whatever was necessary to keep the cooling system going, as a failure would cost hundreds of thousands of Euros. They collaborate with the physicists in order to further develop the system. Usually, the group leader triggers changes that are afterwards realized as projects. Often other companies are involved in their implementation.

At the time of the study, they were engaged in a project to replace parts of their software infrastructure. They aimed at a software suite to visualize sensor data and complex combinations of sensors, which were built into the cooling system. Six people worked constantly on this project at that time. Several employees from Delta were also involved. Sometimes members of Delta visited Epsilon to present the current state of the project or to work collaboratively with the team at Epsilon.

The work on this project is to some extent decoupled, as the architecture of the system allows this kind of flexibility. Meetings to refine requirements, to exchange experiences or to solve problems usually happen ad-hoc. There is no strict separation of work within the group. If new tasks arise, they are assigned according to people's previous experience, work-load and interest. During critical times of development, some of Epsilon's members are involved in pair programming. Once a week they meet to discuss the state of current tasks/projects.

Stephan is a physicist, holding a Ph.D. He has been an employee at Epsilon ever since the particle accelerator was built and has used Eclipse since the group leader decided on its introduction, several years ago. Additionally, he is a member of the workers' council. This means that he can only spend a small amount of time on development and thus still saw himself as an Eclipse beginner. He uses Eclipse mainly to test his co-workers' developments. His role is to negotiate between the developers (his colleagues) and their users (the operators).

Klaas studied computer science and had formal training in electrical engineering. He has worked at Epsilon for the last 3 years. At the time of the study, he was involved in the implementation of Eclipse-based control room software. He had been using Eclipse for 2.5 years at the time.

Heinke studies business economics. He spent half the week working as a Ph.D. candidate and research associate at the local university and the other half at Epsilon, working on the same project as Klaas. He had been familiar with Eclipse for 5 years at the time.

Zeta: Medium sized web development company

Zeta is a medium-sized business, employing about 250 people, comprised of several branch offices throughout the country. Founded in 1991, zeta has since then developed mainly web-based, customer-specific solutions, e.g. e-commerce systems and interactive websites. However, they are also experienced in other areas such as small games, multimedia and research projects. Their customers are mainly large enterprises.

Zetas employees use whatever technology is necessary to fulfill customer requirements. Typical web-based projects generally use PHP or Java for example, although Perl and Ruby are also in use. At the time of the study, Eclipse was used for nearly every project that involved Java as a programming language and furthermore for several other projects that use PHP and other languages.

The work is managed in projects. Usually, projects start with a technologically-experienced person as project leader. This person is responsible for communicating with the customer and for coordinating developers as well as service providers. Although Zeta does not suggest a specific software engineering practice (i.e. agile methods or V-model approaches), the developers and project managers tend to follow an agile approach. This is furthermore also influenced by customers' preferences. Practices such as using a common code repository or using coding conventions are typically part of every project. Ultimately, the specific project team decides how to do the job.

Depending on emerging or expected needs, employees undergo training to develop their skills in using different practices or a specific technology. During project work, developers work together closely, generally exchanging experiences as needed and helping each other should problems materialize.

At the time of the study, the overhead work for managing Eclipse-based working environments was a huge problem for Zeta's employees. They spent too much time trying to freeze every work place so it could be accessed at a later date, following completion of a project. Furthermore, they were using several tools that were so worryingly unstable that they had destroyed the working environments of several people on a weekly basis. One person was therefore allowed to work full time on a provisioning solution to provide Eclipse-based working environments for specific projects. An Eclipse working environment was created, fitted with certain plug-ins and configuration settings, which the project team would later need. At that point, the whole working environment was committed – file by file – into the company's Subversion source code repository. This way, the rest of the project team was able to download it from there. While this approach held some benefits, it also introduced several new problems due to instability, bad usability/user experience, high complexity and low transparency.

We met a team, which was involved with a web project at the time and which had tested the self-made provisioning solution.

Simon is 31 years old. Since 1999 he has mainly developed software. First and foremost he is involved in Java programming for the web. However, he sometimes works within the company as a consultant. He has been using Eclipse since Version 2.1.

Francois is 30 years old. He has been a software developer for 7 years, joining Zeta just a few weeks before commencement of the study. He works with PHP and Java for the web. Furthermore, he is an expert on different Java-related frameworks.

Robin is 38 years old. He has worked as a software developer for the last 7 years and has used Eclipse for the last 4 years. He mainly works with large customers and has maintained some of their systems and projects for several years now. He regularly meets customers (in contrast to Simon and Francois, who don't).

Leo is 35 years old and works as a software developer at Zeta. At the time of the study, he was working on the internal Eclipse provisioning solution and had a lot of contact with his colleagues for requirement engineering and testing purposes.

Theta: Software development and Eclipse consultancy company

Theta is a small/medium-sized German company. It was founded in 1998 and employs about 30 people. When the workload is heavy, Theta employs up to 50 people at times, including several freelancers. At the time of the study, Theta had opened another site in eastern Europe (four people) and was in the process of fusing with a very small american company (two people). Altogether, Theta has the closest connection to Eclipse and the Eclipse community.

Theta's business is based on two columns. The first was a self-developed Eclipse-centered software provisioning solution (ThetaProduct), which was initially released in late 2004. Theta licensed ThetaProduct to large companies and sometimes also provided hosting of the solution as a service. An additional offer Theta made their customers was to develop specific adaptations of ThetaProduct. Theta's second business column was Eclipse-related consulting, especially how to develop business applications on top of Eclipse technology. In this line of work, Theta developed ThetaFramework, which helped to create Eclipse-based web applications. ThetaProduct was based on ThetaFramework and was used as a showcase for ThetaFramework. ThetaFramework was released under a open source license and Theta sometimes makes use of business models like sponsored development, which work best in open source communities.

Theta held experience exchange among the staff to be very important and for this methods such as pair programming were very often used – although not exclusively. Furthermore stand-up meetings allowed the whole team to be brought up to speed every morning. Besides these more formal approaches towards sharing experiences during work time, there was a commonly used coffee-kitchen. People spontaneously met there or went there together to have a coffee and to collaboratively discuss and solve tough problems relating to the tasks at hand. Once a week, one of the employees

gave an informal talk on a topic of his choice. Although these talks were very often work-related, this was not compulsory. New technologies or Eclipse-related aspects were often discussed there.

As ThetaProduct and ThetaFramework are extremely Eclipse-centric, everyone in the company was an exceptional Eclipse expert. This was true for the usage of the Eclipse IDE but also for the development of new applications, based on Eclipse. Most people either worked in the ThetaProduct or ThetaFramework team, as new features had to be implemented and compatibility with the constantly changing Eclipse technology had to be guaranteed. This work is often interrupted to give priority to customer specific work. Usually, the team leaders divided large tasks in smaller ones, which were then distributed to the team members. Tasks were assigned according to the employees' experience and preferences. During the study, we took a closer look at the ThetaProduct team.

Emil is a trained management assistant focusing on data processing. Having worked as a software developer for more than 10 years, he was experienced in a variety of programming languages such as Pascal, Delphi and Java. He has also been an Eclipse user since 2001 and an Eclipse plug-in developer since 2002. For the last 3 years, he has worked for Theta.

Antonio is a trained media designer who specialized in web applications. He has been a member of Theta since 2004, when he started Java and Eclipse plug-in development. He worked part time for the ThetaFramework project and is currently project leader for ThetaProduct. He works with customers and sometimes does consulting work at a customers site or at conferences.

Eugene is a management assistant focusing on data processing. For the last 2 years he worked at Theta as an application developer and currently began studying business informatics. For more than a year he is working as a student for Theta, mostly with the ThetaProduct development team.