Appropriation of the Eclipse Ecosystem: Local Integration of Global Network Production

Gunnar Stevens and Sebastian Draxler

Abstract Eclipse and Mozilla Firefox represent a new type of open software that can be supplemented by manifold extensions, being implemented by independent software vendors and open source projects. Research on such software ecosystems shows that collaboration patterns in the software industry evolve from value chains to value nets. An often ignored side-effect of this development is a vast extent of integration work that needs to be done by users. Taking a user point of view, this paper presents an empirical study on the practices of appropriating the Eclipse ecosystem as an example of radical tailorability, based on new opportunities given by the surrounding ecosystem. We show the practices users have developed to manage the antagonism of maintaining a stable and productive working environment, while simultaneously innovating it. Based on these results, we outline different opportunities to improve flexible software by supporting cooperation among the diverse actors involved, in a network of production and consumption.

Introduction

How do end users tailor and appropriate their computational working environments?

This question has been intensively investigated in HCI as well as CSCW research [inter alia 1, 2–6]. However, studying the tailoring of Eclipse as a daily working environment in an ethnographical manner, reveals a topic that is rarely discussed in literature – namely the fact that the tailoring work is 'shaped by' and 'part of' a software ecosystem.

The term software ecosystem was introduced by Messerschmitt and Szyperski [7]. It is semantically related to concepts such as production networks or network economies, but tries to integrate the economical and the technological point of view.

G. Stevens (✉) and S. Draxler
University of Siegen, Hölderlinstrasse 3, 57068 Siegen, Germany
e-mail: gunnar.stevens@uni-siegen.de; sebastian.draxler@uni-siegen.de

A Software ecosystem can be defined as a network of related actors, interacting with a shared market [8]. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts.

Technologically, software production networks are related to McIlroy's vision of component-based software development [9]. As early as 1968, he saw future application development as plugging together 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. Since the 1990s, a large number of changes have been accomplished to make his vision real. Furthermore, we observe a tendency of supplementing major products in the different market segments by loosely coupled networks of Independent Software Vendors (ISVs) and open source projects, which provide extensions and services to the core product. Organizationally, software production networks are mainly studied with focus on coordination in globally distributed software engineering [10].

Existing research on production networks focused mainly the developer perspective, neglecting the user perspective. But it is the user's role that changed, because unlike McIlroy envisaged, there is no general contractor. Tailoring eclipse may hold as example here. The work of picking the right pieces from a loosely coupled network and integrating these into their local work context are 'outsourced' to the user's sphere of responsibility.

Even Grinter, as one of the few who studied both – global coordination in distributed software production [10] as well as local integration work [2] – missed to draw the connection between both views. We introduce a brief gedankenexperiment to raise the awareness of this topic (see also Fig. 1) and its implications. Assume a world1 where a developer A creates an UML tool extension for the Eclipse platform. Taking a bird eyes view on distributed software production, one can conclude that the work of developer A depends on the work of the Eclipse platform developers. In a world2 a developer B creates a Mobile phone extension for Eclipse, so that we conclude in an analogue manner that the work of B depends on the work of the Eclipse developers. The platform in world1 and world2 is the same, but it does not constitute a need for cooperation between A and B [see also 11].
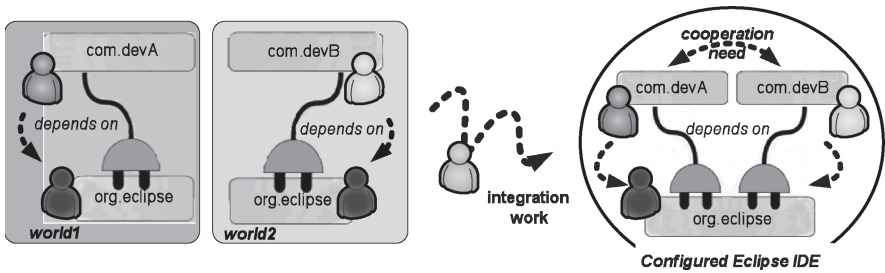

Fig. 1 Constituting new cooperation needs in the actor-network of a software ecosystem, through the integration work of the users

Now, what happens if a user C brings world1 and world2 together by downloading the extension A and B to use both together (e.g. because his project leader have told him that in the current mobile phone app project he should document the architecture concept with the help of UML diagrams). As mentioned by Grinter, user C has to carry out some integration work, which potentially confronts him with the problem of compatibility issues between A and B. In reflection of this anticipated breakdown, one might argue this problem arises because of missing cooperation work between the developers A, B and the Eclipse developers. This is true of course, but neglects the fact that the need to cooperate was introduced by the integration work of the user.

The argument illustrated by the example is that in software ecosystems cooperation is sometimes constituted by the integration work done by the users. In this article we do not investigate the consequences to study cooperation in distributed software development, but using this as a theoretical consideration that shapes our analytic lens in the following manner: Firstly the example explains why breakdowns are no accidental phenomena, but essentially connected to the freedom of choice provided by open software ecosystems. Therefore we need a profound understanding of the practices to integrate a loosely coupled network of components into the local context. Secondly, understanding integration work of the user as part of the distributed development in software ecosystems influences the framing of the design space and motivates to search for new solutions to support the cooperation in the actor network given by the software ecosystem. However, we give only an outlook on this new field of CSCW research, while our primary focus is to explore the practices of users appropriating the Eclipse ecosystem, while being under the pressure of getting their ordinary job done.[1]

Based on this focus, the paper is organized as follows: Section Bringing a Component Network into Practice gives an introduction into the research on designing and managing an adaptable software application. Further, Section Eclipse Workplaces as an Expression of a Global Ecosystem gives a brief introduction into the Eclipse ecosystem, illustrating, how the Eclipse ecosystem functions as a decentralized, open production network that gives users new opportunities to adapt their working environment to the local needs. However, the openness of ecosystems prevents that developers can fully anticipate all the side effects that emerged in the local context. Against this backdrop, Section Managing the Eclipse Ecosystem in Practice presents an empirical study on Eclipse tailoring practices, making use of the dynamical evolving market of Eclipse plugins. We close with an outlook of design options to support end users appropriating software ecosystems.

---

[1]Eclipse users are typically software developers. Despite the fact that this group does not present 'the' typical end user, we decided after discussing the pros and cons, to investigate the appropriation practices of Eclipse. The main reason is similar to a 'lead user'-approach as we can observe emerging strategies to make use of a widespread, dynamical and complex software ecosystem. It has millions of users, it realized a highly advanced technological concept of 'everything is a plug-in' [12] and its complex ecosystem offers one of the most advanced platforms to enable a network economy in the software industry.

Bringing a Component Network into Practice

The CSCW and HCI literature that has taken into account the user perspective, can roughly be divided into research on the creation of end user-oriented flexibility on the one hand, and research on the management of the flexibility of IT-infrastructures on the other.

Design for Flexibility

Tailorability is a concept that allows users to flexibly fit an application into their context of use. Tailorability is a general demand in the concept of End User Development [4]. On the background of Participatory Design, the aim of tailorability additionally is to support the democratization of the workplace [3, 4]. Tailoring takes place after the original design and implementation phase of an application [13]; it typically starts during, or right after the installation of the application in the field.

The scope of possible changes can consequently be rather broad. Henderson and Kyng [3] distinguish three levels of complexity: choosing between alternatives of anticipated behavior, constructing new behavior from existing pieces and altering the artifact (i.e. reprogramming). Several research prototypes have been implemented in the 'design for flexibility' approach. In particular the CSCW research systems have followed more or less a concept of component-based tailorability [14] similar to Eclipse and proposed highly tailorable groupware application frameworks.

Managing Flexibility

In spite of the great extend of research on making systems more flexible, there are only few studies tackling the management of related flexibility in practice [1].

Based on a field study, Bowers emphasizes the unanticipated work required to make flexible systems work [1]. He notes that there is a unique way to deal with this issue and that a significant burden is not always recognized as such in a working environment. Extra work can even be a reason for abandoning technologies or certain courses of action. On the other hand, sometimes "it might be regarded as 'a good job for a junior to do in order to find out what is going on here' and so on and so forth." [1].

Grinter et al.'s empirical studies point out that managing a home IT infrastructure is a collaborative issue, where an informal division of labor arises [2]. Typically, the party with the biggest technical competence gets the job to run and maintain the network. In their field study, Star and Ruhlender observe that 'signing on and hooking up' is a form of artful integration into a socio-technical infrastructure, in large scale as well as in situ. This integration reciprocally shapes the infrastructure [17].

In particular, 'infrastructuring' is a severe and complex issue, where "socio-material relations of multiple, heterogeneous elements and the collective, situated inter-weaving of people, artifacts and processes" [18] inter-relate. Pipek coins the term "shared infrastructure scenario" [16] to describe such a constellation. Balka and Wagner point out that the configuration of a technical infrastructure to make things work is part of the appropriation process [18].

Several empirical studies have demonstrated that tailoring is usually carried out collaboratively by end users, local experts, IT support or helpdesk staff, and takes place in social networks within organizations [15] or within user communities [16]. Based on these results, different authors have suggested cooperative solutions for customization [4]. However, these studies rely on the diffusion of adaptations, which has been created in the local context and not on the adoption of extensions, which are available within a worldwide community.

All these studies demonstrate that managing flexibility is more than simply assembling and configuring components, indicating that tailoring has to manage the complexity of integrating independent, but inter-dependent production processes.

Eclipse Workplaces as an Expression of a Global Ecosystem

The case of Eclipse is in several dimensions an example of a global software ecosystem in recent commercial software production. Each of them is worth being studied for its own sake. But we want to concentrate on three different facets of Eclipse that influence the socio-technical environment of Eclipse users.

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 of how to integrate a heterogeneous network of product development divisions into a global informational production ecosystem [cf. 19], where a distributed development process has to be coordinated [20].

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. IBM Software group had grown rapidly, also by the fact that IBM has 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'), as well as of 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 could not share compo-nents 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. 20] 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 application – 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, were all members of the consortium should 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.*" [19].

Today, Eclipse has become 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 1,000 different extensions and applications on the top of the Eclipse platform) and/or for an ecosystem (where an Open-Source community co-exists with commercial players). In addition, commercial products like ondemand.yoxos.com or poweredbypulse.com are specialized in maintaining repositories of third 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.

"Everything Is a Plug-In": The Technological Fundament of an Ecosystem

Eclipse is a living software ecosystem that faces the problem of a consistent evolution of the heterogeneous network of producers and products. The strategy Eclipse realizes 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 [12] 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 version Eclipse implemented its own component model, but since version 3 it switched to 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's architecture not only creates a dependency graph 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 third 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.'*" [19].

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" [12].

Based on this idea, they argue that the plug-in concept constitutes a pyramid of increasing commitments and rewards, in which the committers of the Eclipse Foundation are at the top. In the middle of the pyramid are publisher and enablers, who contribute third-party plug-ins to the Eclipse Ecosystem without being part of the Eclipse core. End users are also part of the game, as they build the bottom of the pyramid. They can influence the design of Eclipse directly by configuring and extending an Eclipse installation. 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.

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 innovating it simultaneously. One of the major problems in this process is that further development of one piece in the global plug-in network can lead to a crash in another part. The only secure method to prevent this is to stop any changes, but this also hinders 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 in 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.

Discussion

In 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, but a political institution of different interest groups – presents the centre 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.

Managing the Eclipse Ecosystem in Practice

Our empirical study of (collaborative) appropriation practices of Eclipse users was part of a public founded research project (CoEUD), where we cooperated with four different software companies in Germany. Two companies, a groupware producer

(company alpha) and a web-development specialist (company beta) started to experiment with Eclipse as their development environment. Both employ approximately ten people. The third enterprise (company gamma), also a web-development and IT-infrastructure specialist (approximately 250 employees), already used Eclipse as the standard working environment for quite some time. The fourth company (company delta) is part of a holding, while the participating group consists of roughly ten members. This group is specialized in executing eXtreme Programming projects and in providing this expertise to other companies. This company also builds domain specific applications on top of the Eclipse platform, though this was not the primary focus of our empirical study.

These four companies were selected as research partners for different reasons: First of all, they use Eclipse in their daily work practice, in addition, they represent typical enterprises of the German software industry. These are usually classified as Small and Medium Enterprises (SME). Additionally, there already exists a long and trustworthy relationship between the companies and the researchers, which was an important factor doing workplace studies as a part of participatory action research.

In its attempt to improve the flexibility of Eclipse from an end user perspective, the CoEUD project followed the *Business Ethnography (BE)* approach [21], where participatory design and ethnographical informed analysis are of equal importance. *BE* does not draw on direct implications of ethnography for design, but rather on the decomposition of projects into different reference frameworks of participants, based on their working practices and views. This decomposition can be used as a reflection method to support mutual learning and discourse processes between participants of the project. However, in this paper, we focus only on the findings of this empirical study and not on our intervention in the field.

In each company, we conducted at least two semi-structured interviews (altogether, we conducted ten interviews from August 2007 to September 2007). We interviewed four junior developers, four senior developers and two CIOs, all of them were using Eclipse for their daily work (except one CIO). The interviews took about 1 h and covered questions about role, tasks and responsibility of the interviewees in the enterprise. In addition, we asked questions about their experience regarding Eclipse as well as their update and learning strategies.

Additionally, we visited two SMEs (company beta and delta) for a defined period of time (3–5 days) for participatory observation. The observations were accomplished in the typical ambience of the developers, in order to get a detailed insight in their working activities. The participant observations were written down as field notes. Furthermore, we also cooperated intensively with company gamma, which had established a project to develop a prototypical solution to support a company-wide provisioning of Eclipse. In three design workshops with the project leader, we discussed requirements and implementation opportunities for a cooperative provisioning solution. As part of the participatory action research approach of *BE*, we actively participated in these workshops, discussing potential side effects of centralizing the administration of working environments.

All interviews and the workshop were recorded, partly transcribed, paraphrased and analyzed together with field notes and supplemented by personal experiences. In a second step we selected specific parts of the empirical data for a microscopic

examination using the 'Kunstlehre' of sequence analysis as suggested by Oevermann [22]. Similar to Grounded Theory, the aim is to reconstruct the categories from the case instead of subsuming the case under pre-defined categories. Similar to Conversation Analysis it is guided by some interpretation principles like immanent, extensive and verbatim interpretation of record following the sequential structure applying the principle of austerity.

With the help of our qualitative studies, it was possible to understand the work practice and to uncover and document situational work practices and strategies in respect to the appropriation of the Eclipse ecosystem. In order to triangulate our findings we additionally conduct an quantitative oriented online survey from February 2008 until April 2008 and analyze the Eclipse configuration used in practice following a 'mix method' approach [23].

The online survey consisted of a questionnaire, which additionally asked the participants to add certain Eclipse configuration data. This allowed us to analyze which plug-ins have been installed by the online study participants. It was announced in different online forums, mailing lists, by our project partners and 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 aimed at the local context and experiences regarding Eclipse, but also asked for information on the local Eclipse configuration, which gave insights into the features and plug-ins the users had locally installed. One hundred and thirty-eight persons participated in the survey and 59 additionally sent us their Eclipse configuration, which we analyzed in detail.

In the following we present the results of the online questionnaire analysis that draws a first picture of appropriation practices of Eclipse users. Afterwards we show the core findings of the interviews and participatory observations to point out the user's motives and strategies in modifying Eclipse.

The Quantitative Side of Managing the Eclipse Ecosystem

Fifty-nine of 138 participants of our online survey sent us information about their Eclipse configuration. Surprisingly we received 76 configurations for our analysis, because some persons were using more than one configuration. This also means that these users own more than one Eclipse installation on their computer (in our workplace study presented in the next section, we found some reason for that phenomenon).

As a first step, we were interested how many plug-ins are used in practice. This should help us to answers 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 is 326 plug-ins per configuration.

Furthermore, we analyzed the so-called "features" of the captured Eclipse configuration data, as these are the basic elements of update management and configuration management in Eclipse.[2] The concept of features reduces the complexity of the plug-in network for the users. Instead of managing about 300 plug-ins, the user only has to manage around 40 features (cf. Table 1). 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 configurations. 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.[3]

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 third party plug-ins from independent ISVs are integrated into Eclipse installations. We focused therefore on features that are not delivered by the Eclipse foundation. One of these features is the support for the *Subversion* source-code version control system for Eclipse, which was by this time provided by two different independent open source projects. At the time of the survey, none of these tools were integrated into Eclipse by default; instead it is up to user to integrate this extension into the Eclipse installation if Subversion support is needed. In our sample 40% of the Eclipse configurations included Subversion plug-ins, which is a strong indicator that the users make use of the global market of Eclipse extensions.

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

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

---

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

[3]We calculated the distance of two configurations with the set of features Ci and Cj as follows: $u_{feature}(Ci,Cj) = (|Ci \backslash Cj| + | Cj \backslash Ci|)/(|Ci| + |Cj|)$. Based on this calculated the average distance: $\bar{u}feature(C1,\ldots, Cn) = 1/n*(n-1)*\Sigma_{0 \le i < j \le n} U_{feature} (Ci,Cj)$. 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.

In order to learn how the evolution of Eclipse is reflected in the configuration 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. Sixty configurations are of the 3.3.X release (published June 2007), 12 cases of the 3.2.X release (published June 2006) and three cases of the 3.1.X release (published June 2005). We did not find a configuration 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 if 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 of the participants (92.66%) declared they 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 configuration data. In addition, it shows that adapting the working environment is not only a singular, but in most cases a regular activity.

We are also interested in strategies to inform oneself about activities of the Eclipse ecosystem, the role of collaboration and configuration sharing practices. In particular, we are interested, if a local network of Eclipse users exists. Therefore we asked: "*How many of your colleagues also use Eclipse?*" The majority (71.32%) explains that in local environments also other persons use Eclipse (only 4.41% say no other person use Eclipse, 24.27% give no answer to that question). This confirmed our workplace observation that in most cases a local social network of Eclipse users exists.

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 demonstrated that the Internet as a global resource is the most used source for information, but it also demonstrated 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).

We were also interested in the channels used for diffusion of plug-ins, therefore we asked: "*Which ways did you use to receive these plug-ins?*". Figure 2 gives an overview on 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 strong indication that local networks also play an important role by the diffusion of plug-ins, although this was not anticipated by Eclipse designers and it is not well supported by Eclipse.
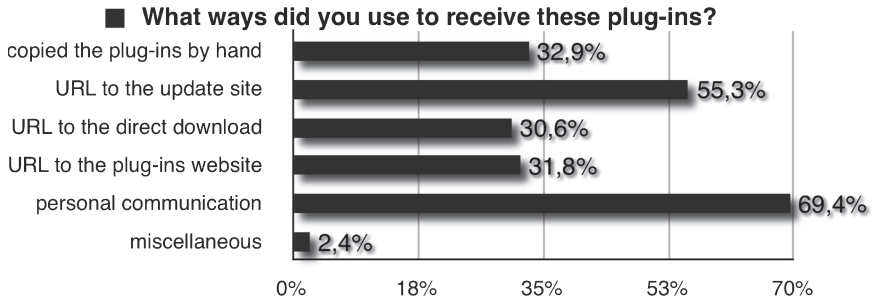
■ What ways did you use to receive these plug-ins?

copied the plug-ins by hand	32,9%
URL to the update site	55,3%
URL to the direct download	30,6%
URL to the plug-ins website	31,8%
personal communication	69,4%
miscellaneous	2,4%

0%   18%   35%   53%   70%

Fig. 2 Channels used to receive plug-ins and plug-in information[4]

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 configuration. It also demonstrates that local social networks play an important role in the appropriation of global Eclipse network of loosely coupled components.

Doing Integration Work

In our workplace study we focused on practices and motives of integrating Eclipse as part of the daily work. Similar to the quantitative data, our observation shows a huge diversity and a highly dynamic evolution of Eclipse installations and the importance of local social networks in appropriating the global Eclipse ecosystem.

From our ethnographic point of view, we identified one reason for the diversity of Eclipse configurations. There are no strict regulations for tool configuration in the observed companies. Furthermore, software development is organized in projects, which are highly dynamic and diverse by themselves. E.g. the company gamma mainly implemented individual non-standard software for their clients' work with project durations ranging from 1 week up to 1 year or longer.

One of the key competences of this company was its knowledge about current technology trends and its competence in building an individual integration solution on the top of heterogeneous IT-infrastructure of their clients. In most projects special tools like PHP or XSLT editors or environments for testing purposes, like databases or application servers are needed. Logically they belong to project-specific

---

[4]The other way round *"Did you ever share plug-ins with colleagues?"* and *"Which way did you use to share these plug-ins?"* provide a nearly identical pictures.

working environments, but technically they cannot be well integrated into the management of the Eclipse working environment.

Typically, a project at gamma starts with one or two developers, but during its lifetime the core team sometimes calls upon further developers as experts for specific technologies or tasks (e.g. an expert on a particular database or an UI designer). As a consequence, the consulted experts have to synchronize their working environment with the one of their colleagues in order to cooperate with each other.

This means, that the project context and progress triggers the modification of Eclipse, but nevertheless leaves room for negotiation and individual exceptions from project norms if this is needed to get the work done. In addition, the individual appropriation of new Eclipse plug-ins can also become the source of grass root innovation, where one picks up a new feature that then diffuses into the whole organization through local social networks e.g. through the shared working context of project team. This is very similar to Mackay's observations [15].

This way projects establish the context of implicit-knowledge sharing which goes along with the sharing of Eclipse configuration. E.g. the following transcript demonstrates this practice as part of a cooperative appropriation strategy: *"If a new colleague starts to work here, I would advise him to begin with the standard IDE, and as a starting point for further exploration, I would show him which plug-ins... I did integrate in my Eclipse installation... could be of interest for his job (based on his experience)." (Interview transcript with a senior developer at gamma).*[5]

Through the specific characteristics of Eclipse, it is also possible (nevertheless cumbersome) to share best practices through *Copy and Adapt*-installation. Mackay describes a similar phenomenon in the case of customizations [15]. Applied on this case it means, one user sets up and adapts an Eclipse installation and afterwards shares it with his or her colleague(s) in the project: *"I did configure Eclipse a few times. Or rather we did this more or less together. For example [within the project] we use the CheckStyle component and some other plug-ins, of which I forgot the names, because in fact my colleague did set up the original configuration. And I eventually copied the whole configuration over to my workstation." (Interview transcript with a junior developer at company delta).*

Another important aspect is related to the *habitus* of software professionals, who usually consider their own work to be a craftsmanship rather then assembly-line work as Strübing [24] describes his observations. In our study we found that sophisticated knowledge about tools and an *always be up to date* attitude knowing the current technology trends seems to be a core element of this *habitus*. This is similar to the result of Day, who has analyzed the failure of the tayloristic CASE tool paradigm in practice: "Each user [of a CASE tool] has his or her professional

---

[5]This citation also shows how subtle the balance between autonomy and cooperative integration is constructed: The seasoned developer submits an individualized offer, but respecting the autonomy and the tool competence of the other, even if he is a novice. Below, we discuss this topic in more detail.

perspective regarding appropriate models and procedures for software development. Many feel that their trade is as much an art as a science and resist the application of constraints." [25].

Although tool autonomy is an important issue for software professionals, whether something is disrupting the own autonomy depends on the perceived rationality, an often quite subtle issue:

> E: Which Eclipse version do you use?
>
> I2: Mhhh, if it were up to me, I would probably change my Eclipse at least every half a year. Because there …err … happen a lot of things. Though I can also understand when my company says that it is of course a little risky … err …as if bugs show up or incompatibilities … err … Look, I once even wanted to install Eclipse 3.2 here. (laughs) … err … they said the same about Eclipse 3.1. [not to change too often], think that I can really understand. (Interview transcript with a senior developer at gamma)

We can interpret this passage in two different ways. First, we can argue that the developer was bent by the power of the company leading to an alienation from his production means, but this related to the issue that the developer accepts the rules of the game, because he understands the need for the rules of this social context. Regarding the related actor network, the user draws a connection between his company and Eclipse (treating developer and artifact as a unit) and tailoring his working environment is related to manage the conflict in the two different innovation rhythms of this both actors.

Beyond this outline of some triggers for workbench modification, the analysis exposes also particular fears concerning tool modification like a decrease of efficiency or a complete failure of the tools being used. This leads to the antagonism of stabilization versus innovation (or informally spoken "never change a running system" versus "being up-to-date"). This antagonism reproduces the general characteristics of the evolution of Eclipse, where stability of the environment is very important, but in the area of software development everything is oriented towards innovation, which means that if you don't move forward you go backwards.

As a result, Software developers are aware of new technologies and trends. For example, the field study at company beta shows that the tracking of technology trends was as much a common habit like reading newspaper:

> 9:00 a.m. the developer C sit at his PC workplace, organizes his daily work, e.g. checking his mailbox. During these activities he takes time for visiting bookmarked web pages to read news and trends related to the Eclipse IDE. When asked, the developer explained: '… from time to time I go through the web to see what is new.' (Observation diary beta p. 38).

Also software professionals communicate and discuss new trends and new tools they found on the Internet. This discussion can be a trigger not only to speak about the new tools, but also to try them out.

On the other hand, each modification has to be appropriate and might cause trouble for the working routine. This means that it is not to be taken for granted that every modification is a step forward. Therefore, the developer has to balance the antagonism by taking different aspects into account. For example:

E: Did you always use the newest Eclipse Version?

I1: Well, most of the time only up to the release version, err to the real one[6]

E: err.

I1: Except when there is something special, such as – for example the web tool platform … err … which contains so many features one needs. That it is a real benefit, to really use some release candidates or former versions. Something that's still more or less stable, otherwise the risk is just so high that you could lose half a day or something like that, just because you still somehow... some function, yet again... or something that is still too error-prone (Interview transcript with a senior developer at beta)

Another case demonstrates that the balancing of the antagonism is not an up-front and fixed decision, but can be changed, based on the actual experience of adapting the workbench:

Software developer C tried to integrate a new component into eclipse. During this activity a mistake suddenly occurred, so the system showed a message box including a hint for a version conflict. The developer cancelled the installation with the quotation: 'That's too hazardous... afterwards something might get broken.' (Observation diary beta p. 70)

One can see that in the practice of tool modification the antagonism between "never change a running system" and "being up-to-date" cannot be conciliated for once and for all, but the reasons for one or the other have to be balanced constantly according to the specific context.

Aspects in Managing a Working Environment

In this section we want to describe the tasks of managing an Eclipse working environment in more detail. In particular, we observed that users have developed their own strategies to deal with this issue. For example, a senior Java developer at alpha, who was continuously looking for new features and technologies, created his own strategy for dealing with the antagonism of stabilization and innovation. In the past, he spontaneously integrated new components into Eclipse, which sometimes led to annoying damages and total breakdowns of his working environment. Based on these experiences, he changed this practice. Now he uses different Eclipse installations: Before he changes his working environment, he creates a backup copy to

---

[6]At first glance the expression "the real one" sounds strange. But a fine-grained analysis showed that he applies a conservative release policy (in respect to his colleague) using only major releases. His colleagues also use the innovation-oriented milestone builds, but as they are not that stable, they are "unreal". We could validate this interpretation by using other interview passages. The case shows that Eclipse is updated quite often, which confirms the quantitative data (although the quantitative data also indicates a more 'conservative' or 'rational' update behavior, as we found no milestone build configuration in any of the 75 cases).

generate a fallback system.[7] After an adequate period of testing, he copies the fallback system to a folder containing older installations, which are out-dated, and then uses the testing environment as his new productive system.

Through a careful analysis of the empirical data we found three different aspects of appropriation work, that are related to each other and that we want to show in more detail:

Keeping Workbenches Up-to-Date

This aspect includes all technical issues of installation, updating, configuration and disposing components or component sets. Eclipse addresses different aspects of workbench modification by different user interfaces. The functionality is split into the *"Search for updates …"* dialog for updating existing tools, the *"Search for new features …"* dialog to enhance the workbench and the *"Product-Configuration"* dialog to inspect, update or dispose existing features. A user eager to inspect his workbench environment at the plug-in level has to use another user's interface. If he wants to dispose a specific plug-in, he has to jump to the file system.

So, even if it makes sense to separate these different aspects analytically, the separation on the user interface level leads to several usability problems. The fact that the user can only inspect his environment at the feature level and not at the plug-in level is also problematic, if system malfunctions appear. Also an appropriate support configuration sharing in the local context facing the issue of autonomy and cooperative integration is missing in Eclipse.

Keeping Tool Competence Up-to-Date

Installing a tool is one thing. Being aware of the tool and evaluating it with respect to the own working practices is another. And learning how to use the tool efficiently is a third one. In addition, from an analytical perspective one should separate the aspects of switching to another version from integrating a new tool. The integration of new components or, more specifically, switching to a new release is motivated by the ambition of being up-to-date. In relation to Eclipse, this has two different meanings. On the one hand, the developers in our study always used the newest version of installed components. Concerning the aspect of tool competency, switching to another version of an existing tool is mainly done to refresh one's knowledge. Quite another task is to integrate and try out special third-party components in the Eclipse framework. Here, the modification serves as 'enhancing tool competency'.

---

[7] Here, the qualitative data explain what the quantitative study suggested, namely that some participants have more than one Eclipse configuration. Another reason is that developers sometimes working in several projects, where a different set of tools are needed.

Disposing Workbenches

It seems a surprising fact that 'removing software' has to be taken into account as an own activity, because nothing seems to be easier than to delete a file or a folder. But from a work-practice perspective, the case is not that simple. This is due to two different reasons: This first reason is that some developers are always skeptical about the reliability of the workbench. Therefore they do not just remove outdated installations, but keep different old versions. E.g. in context of Eclipse, the *"Configuration Manager"* feature is responsible for the support of such kinds of functionality. But we observed that users developed their own strategies to deal with this problem. This might be an indicator for the functionality and/or the usability of the solution to be insufficient. The second reason is that the dynamics of projects have to be considered: as each project uses its own specific workbench with tools and tool versions, even if it is finished, it could be opened again, e.g. to fix a bug.

Appropriation as a Collaborative Effort

Another interesting aspect that is already well described in the literature could also be found in this setting. Users collaborated 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 were necessary. We could especially observe collaboration to introduce experienced Eclipse users into a new team or novice Eclipse users into the Eclipse technology. We could find people acting as source of innovation for the whole team, by being its connection to the global community.

The complexity of the Eclipse technology as well as the complex integration of local user-groups into the huge Eclipse community resulted in regular collaborative appropriation efforts. This differs from existing research through the integration of the group into the global community or Eclipse ecosystem.

Discussion

Eclipse is one of the most advanced technologies and software ecosystems today. It allows end users to create an integrated working environment by assembling components from different vendors. Through shifting this work from the manufacturer to the end user, the integration work becomes an important piece in distributed software production.

Our case study demonstrates that this is not just a theoretically given option. The qualitative as well as the quantitative findings demonstrate that users started to appropriate the new opportunities given by the Eclipse ecosystem and take advantage of an open market of Eclipse components. However, our study also demonstrates that following the market of extensions and integrating components from different manufactures into the local working environment is a complex task – even for very

experienced users as in our study, who also, as part of their job, take care of the tools themselves. So, we have to conclude that the openness of software ecosystems in practice is two-edged: While it introduces a new freedom, it also comes with new burdens as the user has to ensure the quality and compatibility of components himself. We should not misinterpret this as a necessity to step backwards to a time, when the integration work was fully done by the producer. Instead, different observations tend to ask for techniques and tools to support this kind of integration work.

In order to explore and systematize the diverse design opportunities, we pick up the train of thought of the introduction. As mentioned, the integration work constitutes new relationships in the actor network of a software ecosystem. Breakdown situations while tailoring Eclipse often reveal an absence of cooperation among the actors involved in the situation, but also create a ground for future cooperation among them. Regarding the opportunities of computer support to manage the evolving cooperation needs and cooperation conflicts, we can distinguish a micro, meso and macro level.

The *micro level computer support focuses* on supporting the individual users to manage their working environment, making use of an open component market. Eclipse already includes some advanced concepts to support the user. First of all, Eclipse implements a software ecosystem friendly component-based architecture. In addition the update and configuration manager (see also Section Keeping Tool Competence Up-to-Date) provide a built-in mechanism to download and install components and to manage the environments configuration. Moreover, the update mechanism does not just install new versions as they get available, it also links the personal software evolution rhythm to the evolution rhythm of other actors in the network, raising the need synchronize these rhythms (see Section The 'Eclipse Way': The Rhythm of Evolution as well as related view of users on this topic described in Section Doing Integration Work). Although Eclipse already includes support mechanisms, the observed workarounds (realizing e.g. backup strategies, work with multiple configuration etc., see Section Managing the Eclipse Ecosystem in Practice) indicate room for improvements.

The *meso level computer support* focuses on groups and organizations, taken into account that tailoring is a social activity where actors rely on their local social network. In the case of plugin sharing we found similar patterns as Mackay [5] described in a study on sharing of customizations. With one important difference: Mackay's customizations where created and shared within a group or organization, while Eclipse plugins usually are created by people outside the local network. We observed that the diffusion or sharing is often rooted in personal contacts, project teams, work groups or maybe even a whole organization (as observed in some of the SME). Furthermore, we observed that such meso level cooperation can even cross organizational boundaries, where project teams with e.g. external consultants become innovation drivers. One important result of our qualitative study was to reveal the connection between plugin integration and specific software engineering expertise (like Test Driven Development) and related to this, the co-diffusion of tailoring/use expertise and the tailored working environment. We observed a lack of tool awareness is correlated to a lack of local expertise awareness. Taken our

empirical results into account, to respect the individual tool autonomy, but support the cooperation in team, we developed a first prototype called Peerclipse [26]. It is integrated into the working environment and establishes a local peer-to-peer network. It retrieves the Eclipse configurations and uses this information to support awareness of available components among the members of an organization. Furthermore it adds a component sharing tool to use the local network as group repository and visualizes (usage) expertise.

The *macro level computer support* focuses on the software ecosystem as a whole, taking into account that tailoring is an integral aspect of the distributed software development. Apart from a few exceptions, like Dittrich [27], existing research on tailoring mainly focuses on individual persons or group collaboration. However, our case study demonstrates that several problems keeping work environments up-to-date are linked to the macro level of open software ecosystems. Therefore the macro level presents a new and relevant dimension in designing support systems.

Through this new dimension the micro and meso level do not become obsolete. Instead, we need good design concepts to connect and integrate the different levels. Although we are just beginning to explore design concepts on this level, we assume that online market places will be a highly relevant space to mediate cooperation needs among the diverse actors of the ecosystem. In particular, products like yoxos.com or poweredbypulse.com started to integrate public market places into Eclipse configuration tools. This is a first indicator of a transformation of the traditional function to mediate between an open network of producers and users, providing additional services like reducing complexity in offering products and finding producers. Beyond this, combining configuration support and mediation functions, allows the mentioned solutions to support the integration work by giving e.g. feedback if the selected components are compatible to each other. Moreover, exploiting the new opportunities of the Internet as a collaboration infrastructure and distribution infrastructure for digital goods, such platforms might play an important role in visualizing implicit or latent cooperation needs, opportunities and in mediating between the integration work made by the users and the further development work of the producers.

However, the role of these agents to enable new forms of collaboration in software ecosystems just in the beginning and further research and development is needed to synthesize the different level of computer support in an appropriate manner.

Conclusion

In this paper we studied the appropriation of the Eclipse software ecosystem, which is constituted by heterogeneous networks of independent but interdependent components and related stakeholders. We should be careful in generalizing our results, not just because of the fact that software developers are trained to solve technical problems, but also it is part of their habitus to follow technological trends.

However, we assume that tailoring applications by making use of software ecosystems will become an important issue in general. Therefore, we should

explore in detail the practices to cope with the antagonism of stabilization and innovation. Also in this open and loosely coupled software production, the role of users who establish (collaboration) relations between different actors has to be further explored. In this paper we identified opportunities to support these activities at different levels. However, these opportunities have to be further elaborated in future. Also a more elaborated concept of cooperation is needed that capture the fluid cooperation constellations in software ecosystems given by the dialectic of production and consumption. This underlines the need to rethink software development in terms of Marx "*Just as consumption gave the product its finish as product, so does production give finish to consumption*".

References

1. Bowers, J. *The Work to Make a Network Work: Studying CSCW in Action*. in *Proc. of CSCW '94*. 1994: ACM Press, New York.
2. Grinter, R., et al. *The Work to Make a Home Network Work*. in *Proc. of the ECSCW'05*. 2005.
3. Henderson, A. and M. Kyng, *There's no place like home: Continuing Design in Use*, in J. Greenbaum and M. Kyng (eds.), *Design at work*. 1991, Erlbaum, Hillsdale, NJ. pp. 219–240.
4. Lieberman, H., F. Paternò, and V. Wulf, eds. *End-User Development*, 2006, Springer, Dordrecht.
5. Mackay, W. *Patterns of Sharing Customizable Software*. in *Proc. of CSCW'90*. 1990.
6. MacLean, A., et al. *User-Tailorable Systems: Pressing the Issues with Buttons*. in *Proc. of CHI 90*. 1990: ACM Press, New York.
7. Messerschmitt, D. and C. Szyperski, *Software Ecosystem*. MIT Press, Cambridge, MA, 2003.
8. Vasilis, B., J. Slinger, and B. Sjaak, *Formalizing software ecosystem modeling*, in *Proc. of 1st international workshop on Open component ecosystems*. 2009.
9. McIlroy, M.D. *Mass produced software components*. in *Software Engineering - NATO Science Committee Report*. 1968. Garmisch, Germany.
10. Herbsleb, J.D. and R.E. Grinter. *Splitting the organization and integrating the code: Conway's Law revisited*. in *Proc. of ICSE'99*. 1999.
11. Schmidt, K. and L. Bannon, *Taking CSCW seriously*. JCSCW, 1992. **1**(1): pp. 7–40.
12. Beck, K. and E. Gamma, *Contributing to Eclipse: Principles, Patterns and Plugins*. 2003, Addison-Wesley, Boston, MA.
13. Muller, M.J., J. Haslwanter, and T. Dayton, *Participatory Practices in the Software Lifecycle*, in M. Helander, T. Landauer, and P. Prabhu (Ed.), *Handbook of HCI*. 1997, Elsevier, Amsterdam. pp. 255–313.
14. Mørch, A., et al., *Component-based technologies for end-user development*. Communication of the ACM, 2004. **47**(9): pp. 59–62.
15. Mackay, W. and L. Angeles. *Patterns of Sharing Customizable Software*. in *Proc. Of Conference on Computer-Supported Cooperative Work*. 1990.
16. Pipek, V., *From tailoring to appropriation support: Negotiating groupware usage*. 2005, University of Oulu: Oulu.
17. Star, S.L. and K. Ruhleder, *Steps toward an ecology of infrastructure: Design and access for large information spaces*. ISJ, 1996. **7**: pp. 111–134.
18. Balka, E. and I. Wagner. *Making Things Work: Dimensions of Configurability as Appropriation Work*. in *Proc. of CSCW 2006*. 2006: ACM Press, New York.

19. O'Mahony, S., F.C. Diaz, and E. Mamas, *IBM and Eclipse* 2005. Boston, Harvard Business School Press, pp. 906–1007.
20. Grinter, R.E., J.D. Herbsleb, and D.E. Perry. *The geography of coordination: dealing with distance in R&D work*. in *Proc. of GROUP '99*. 1999.
21. Stevens, G. and B. Nett, *Business Ethnography as a research method to support evolutionary design.* Navigatoren, 2009. **9**(2).
22. Oevermann, U., et al., *Structures of meaning and objective Hermeneutics*, in Meja, V., Misgeld, D. and Stehr, N. (Eds.), *Modern German sociology,* 1987, Columbia University Press, New York. pp. 436–447.
23. Tashakkori, A. and C. Teddle, eds. *Handbook of Mixed Methods in Social and Behavioral Research*. 2003, Sage Publications, Thousand Oaks, CA.
24. Strübing, J., *Arbeitsstil und Habitus – zur Bedeutung kultureller Phänomene in der Programmierarbeit*. 1992, Universität Kassel, Germany.
25. Day, D., *Behavioral and perceptual responses to constraint management in computer-mediated design activities.* EJC/REC, 1993. **3**(2).
26. Draxler, S., H. Sander, P. Jain, A. Jung, and G. Stevens, *Peerclipse: Tool Awareness in Local Communities*. In Supplementary Proceedings of the 11th European Conference on Computer Supported Cooperative Work. Vienna, Austria, 2009. p.19.
27. Dittrich, Y., S. Vaucouleur, and S. Giff, *ERP Customization as Software Engineering.* IEEE SOFTWARE, 2009. **26**(6): pp. 41–47.