# Appropriation Infrastructure: Supporting the Design of Usages

Gunnar Stevens, Volkmar Pipek, and Volker Wulf

University of Siegen and Fraunhofer FIT

**Abstract.** End User Development offers technical flexibility to encourage the appropriation of software applications within specific contexts of use. Appropriation needs to be understood as a phenomenon of many collaborative and creative activities. To support appropriation, we propose integrating communication channels into software applications. Such an appropriation infrastructure provides communication and collaboration support to stimulate knowledge sharing among users and between users and developers. It exploits the technological flexibility of software applications to enable these actors to change usages and configurations. Taking the case of the BSCWeasel groupware, we demonstrate how an appropriation infrastructure can be realized. Empirical results from the BSCWeasel project demonstrate the impact of such an infrastructure on the appropriation and design process. Based on these results, we argue that appropriation infrastructures should be tightly integrated in the application using the IT artifact itself as a boundary object as well as a bridge between design and use.

## 1 Introduction

We interpret the appropriation of information technology not as a phenomenon that somehow happens once a software application is in its 'application field', but as a network of activities that users continuously perform in order to make a software 'work' in a new work environment, shaping the artifact as a material as well as a meaningful object. Existing practices evolve and result in new practices. Technical flexibility to redesign the application according to specific local needs play a major role in enabling appropriation work. Appropriation work may lead to software usages that go beyond what has been envisioned by the designers of the software application [cf. 29]. It is a specific part of an IT artifact's usage, but it remains also linked (through the artifact's materiality) with its design process and the designer's work environments. Appropriation work needs to be understood as a core concept in the field of End User Development (EUD).

To deal appropriately with the combined efforts of users and designers to successfully establish a software tool usage that satisfies the needs of practice requires a fundamental shift in perspectives on the concepts of 'design' and 'use'. If the target of a design process is not a technology/software/tool, but a certain *usage* (that is stimulated by a certain new technology/software/tool), traditional notions of design processes and product structures become problematic. When does usage design start, when does it stop? Is it a continuous or a discrete process? Who initiates 'design' phases – the developer side or the user side? For which parts of designing a usage are professional designers responsible, and for which parts the 'users' (they may be considered as professional usage designers just with a different expertise profile)? Which competencies and experiences are necessary to perform certain activities of appropriation work?

We see the cracks in the idea of a strict separation of design and use spheres everywhere in practice: In the necessity for software development in cycles, in the frequent software updating procedures, in continuous helpline support provided by software manufacturers, in the differentiation of user roles (scale between end users and power/lead users), in software development contract structures that include 'maintenance', in the practice of user forums in the Internet (that may have been provided by software manufacturers, but also third parties), and also in scientific conceptualizations e.g. with regard to 'tailoring' functions that support design-in-use ([15] and many others), with regard to integrating users into software design (e.g. [12]), with regard to professionalization structures in design and the problems they may cause (e.g. [38]) or with regard to the integration of user-driven innovation in (re-)design processes (e.g. [42]). In fact, the blurring notions of design and use spheres point towards collaboration necessities and opportunities which, we claim should become a central research area in the field of End User Development.

We will first connect our perspective to the scientific discourse in HCI and SE. Based on the perspective of usage design, we will describe a framework for an appropriation infrastructure that allows to bridge between design and use by supporting user-user- and user-designer-collaboration in usage design. We have implemented a first example of an appropriation infrastructure when designing the BSCWeasel groupware. To evaluate the utility of appropriation infrastructures, empirical results from the BSCWeasel project are presented.

## 2 Appropriation Work and Technical Flexibility

We now describe in more detail what we see as relevant aspects of appropriation work. We then discuss how the HCI and SE research communities tackle the issue of technical flexibility. Both disciplines understood the need for flexibility on different levels, a product-oriented perspective can be typically found in HCI discussions, and a process-oriented perspective is typical for SE. We will argue that a linkage between these approaches is essential.

### 2.1 Appropriation Work

Several case studies have investigated appropriation processes of IT artefacts in a long term perspective [17, 25, 27, 31, 40, 43]. They offer empirical insights into the appropriation activities and the resulting changes in work practices, and they also showed that a significant part of the work being done to make software applications work is collaborative. Based on these studies, in [29] we lined out opportunities for collaboration support: (1) articulation support (support for technology-related articulations - real and online), (2) historicity support (visualize appropriation as a process of emerging technologies and usages, e.g. by documenting earlier configuration decisions, providing retrievable storage of configuration and usage descriptions), (3) decision support (in a collaborative appropriation activity, providing voting, polling, etc.), (4) demonstration support (provide communication channels to demonstrate usages from one user or a group to another user or a group), (5) observation support (support the visualization of – accumulated, anonymized - information on the use of tools and functions in an organizational context), (6) simulation/exploration support (show effects of possible usages in a exemplified or actual organizational setting, maybe allow configuration manipulations in a sandbox), (7) explanation support (explain reasons for application behavior, automated vs. communication with experts), (8) delegation support (support delegation patterns within configuration activities), and (9) (re-) design support (feedback to designers on the appropriation processes). This list focuses on user-user-collaboration, and most support ideas still remain challenges that have to be met with appropriate technological support.

But *when* is appropriation work? Orlikowski and Hofman [28] focused on the types of work that are not closely related to the new technology, but that rather result in organizational changes. In their conceptual model to classify organizational changes resulting from the appropriation of collaborative infrastructures, they distinguish three cases: *anticipated*, *opportunity-based*, and *emergent* changes. Anticipated changes are organizational transformations, which can be planned and implemented purposefully when the technology is introduced into the organization. The corresponding appropriation work activities can be most likely anticipated, support may be easily provided. Opportunity-based changes occur spontaneously, but can be planed once the opportunity is clear. It may be hard to estimate occurrence, duration and intensity of appropriation work activities here, but once the opportunity becomes clear, the necessary support may also be obvious (see [31] for an example of a group of users that integrated a technological functionality into their practice in an innovative, unforeseen manner). Emergent changes happen spontaneously, and when they happen, they also can't be planned or anticipated, they show that there is a necessity to continuously provide easy access to a broad variety of means for appropriation work (see [41] for an example of creating spontaneous learning opportunities between end users).

There are several approaches that address spontaneous change activities, such as help systems, exploration environments, user hotlines, or the general technical in-use flexibility of tools [cf. 43]. However, these approaches are fragmented and do not refer to each other conceptually and technically.

## 2.2 Product-Oriented Flexibility

The HCI community, and the EUD community in its mainstream, regards technical flexibility mainly as a product feature which allows tailoring computer applications within their contexts of use [15]. Tailoring takes place after the original design and implementation phase of an application; it typically starts during or right after the installation in its field of application. Tailoring is usually carried out by ordinary users, local experts, system's support or helpdesk staff in a collaborative manner. The users may find themselves confronted with technical flexibility on three levels of complexity: (1) choosing between alternatives of anticipated behavior, (2) constructing new behavior from existing pieces and (3) altering the artifact (i.e. reprogramming).

Highly tailorable software artifacts have been developed, commercial products (e.g. spreadsheets and CAD systems) as well as research prototypes ([22], [23]). With the emergence of collaborative tool infrastructures that support communication, cooperation, and knowledge exchange, the need for tailorable software artifacts even increased [3, 46]. The distributed nature of these systems and the potential interrelation of individual tailoring activities posed new challenges to the design of tailorable applications [26]. OVAL [21], Prospero [7], and FreEvolve [37, 47] answered this challenge by providing highly tailorable groupware application frameworks grounded in different paradigms of software engineering.

However, offering technical flexibility is not enough, we also need methods to find the right kind of flexibility to address the requirements of particular contexts of use, considering that things may change over time. It is a short coming of the EUD discussion around tailorable systems that the approaches address the issue of flexibility on the product level only, and do not study how their products are related to the appropriation dynamics and process-oriented flexibility.

### 2.3 Process-Oriented Flexibility

With the idea of designing usages, the traditional design work extends into the use phase. Requirements for tailoring functions can hardly be foreseen completely; inevitably breakdown situations will appear which cannot be fixed using the given tailoring possibilities. Therefore, the development of tailorable software should remain connected to a flexible software development process. The development process needs to be organized to cover rather spontaneous requests for software revisions, as well.

The STEPS software development process model by Floyd [12] extends earlier models of iterative software development with a stronger focus on user-designer collaboration and the gathering of actual use experience (not only laboratory evaluations) in the process of refining software. However, it remained pretty abstract and rather unspecific with regard to the types of work that would be required in addition to ‘programming’ and ‘using’. As a pre-WWW approach, it did also not address issues of collaboration support. An extension of STEPS towards remote participation was suggested in the CommSy project [8]. CommSy uses its own groupware functionality to allow dedicated end users to participate remotely in the design process. Wulf and Rohde [46] proposed, as a part of their OTD approach, to enhance the STEPS model by integrating tailoring as a use activity with design relevance. An implementation architecture for component-based tailorability has been developed within the FreEvolve project [37, 47]. However, the approach did not address the underlying software development process or any necessities to support appropriation processes. Some of these issues were addressed in the concept of a use discourse infrastructure [29].

Coming from a different angle, Fischer discussed end-user modifiability for general design environments [9]. In his early work, he chose approaches similar to other product-oriented concepts dealing with flexibility. Later he developed a design process model called SER (Seeding, Evolutionary Growth and Reseeding; [10]). Similar to STEPS, the concept does neither describe the collaborative work tasks necessary to perform the process, nor does it specifically address the issue of supporting these tasks. The different work on end-user modifiability and participatory oriented design are currently integrated on a conceptual level by the Meta-design framework [11].

Agile software processes, like SCRUM or eXtreme Programming (XP), provide extreme short release cycles and allow customers to change requirements at any time during the design process. In particular XP suggests several methods to make such changes economical and technological feasible. Major software engineering approaches to ensure this type of flexibility are test driven development, continuous integration and continuous refactoring [2].

eXtreme Programming suggests that there should not be any extra effort to fulfill requirements that may appear in the future. This is the counterpart to the concept of radical tailoring. Radical tailorability wants to solve the problem of non-anticipated requirements by building highly flexible products, XP wants to solve this problem by providing flexible processes.

Since XP does not care about the appropriation processes in the use context, the model does not make any suggestion about a shared infrastructure to foster mutual learning processes. Instead, the programmers just get indirect feedback mediated by the "costumer on site"-principle [2], although in practice it is difficult to find these customers [32].

A pragmatic application of agile methods is offered by the development process of the Eclipse platform, called the Eclipse way [13, 20]. The Eclipse way is a mediating position between the dogma of radical tailorability and eXtreme Programming. It follows the position of radical tailorability since all development is based on components to keep the software adaptable and extendable in order to deal with the heterogeneities and dynamics in the fields of application. However, the Eclipse way does not assume that product-oriented flexibility will fully solve the problem of future requirements. It has developed some techniques to foster feedback from users. However, Eclipse does not provide a technical infrastructure to bridge the gap between designers and users. Moreover, the Eclipse way has never been applied to domains in which users are non professional designers.

## 3 An Infrastructure for Appropriation Support

The state of the art does not provide technical support for appropriation work from a 'design for usages' perspective. To fill the gap, a technical infrastructure for user-user and user-designer collaboration is proposed here. The design of the infrastructure is based on two basic assumptions:

(1) Appropriation processes require knowledge sharing among users. Therefore, communication channels should support communities of users to reflect upon the usage of their software.

(2) Support for appropriation processes needs to bridge between product- and process-oriented flexibility. Therefore it is necessary to provide communication channels between users and developers.

Figure 1 illustrates our model of an appropriation infrastructure. We assume that different users, power users, or system administrators work with a software application that is assumed to be flexible in a product-oriented sense. In an EUD sense it could consist of software modules which are represented at the user interface, are meaningful to the users, and can be tailored by them. The communication channels

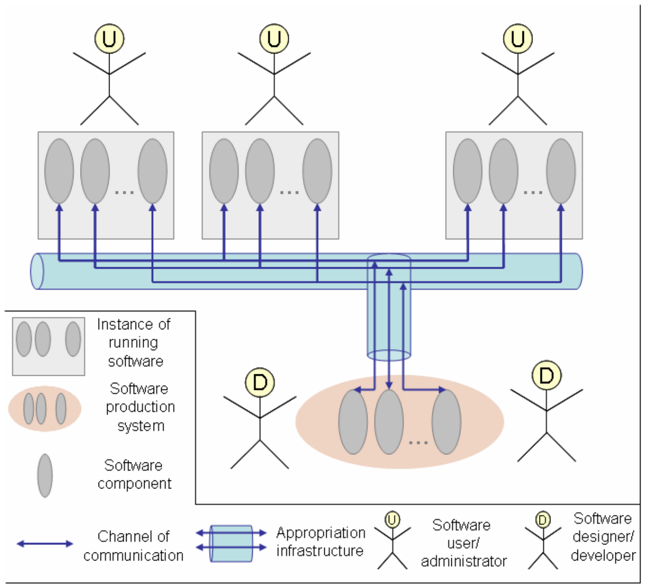

U
U
U
D
D
Instance of running software
Software production system
Software component
Channel of communication
Appropriation infrastructure
Software user/ administrator
Software designer/ developer

**Fig. 1.** Infrastructure for Appropriation Support

among users should hence be integrated into the user's interface and refer to the modularized structure of the functionality (see section 5).

Moreover, our approach is based on the assumption that a team of designers and engineers deal with the modularized code for maintenance and redesign purposes. Communication channels should allow users to express design requirements towards the software team referring to the modularized structure of the user's interface (see section 6). The content of the functionality-related communication among users should become visible for designers as well.

To act as a boundary object among users, the functions of the application and their tailoring options need to be understandable from different backgrounds of practice and levels of expertise. Developers should be equipped with support that enables them to perceive the usage of the application and to recognize break-down situations. Moreover, they need tools to efficiently provide additional flexibility, implement changes or refactor an application. We believe that access to a repository of components could contribute to more efficient work processes.

## 4 BSCWeasel

To explore the idea of an infrastructure for appropriation support, we have developed a groupware application, BSCWeasel, which contains communication channels for appropriation support.

BSCWeasel is a rich client based on the BSCW platform. BSCW (Basic Support for Cooperative Work) was one of the first web-based groupware applications. It was developed at the German National Center for Research in Information Technology (GMD) during the mid 90s [4]. It offers a 'shared workspace' which supports a group of users to up- and download documents. Additionally, awareness services, differentiated access rights, a group management tool, email distribution lists, a discussion forum, and a shared calendar complement the functionality of the groupware.

The fully web-based solution of BSCW has specific advantages. Obviously, there is not any installation effort on the client side. However, there are also considerable technical limitations due to the fact that BSCW just offers a thin client. There is not any redundant local storage for important files, a permanent internet connection is required, and streaming information (e.g. to provide peripheral awareness) is difficult to implement.

Therefore, we have developed a rich client extension, called BSCWeasel which is based on Eclipse. BSCWeasel started as an open source project in spring 2004 (cf.: http://www.bscweasel.de). So far we still follow the basic client server architecture of BSCW where the clients interact with a BSCW server. To implement rich clients, we used the component-based software development environment Eclipse Rich Client Platform (RCP) as the application framework [34, 35].

Eclipse is a development environment for component-based applications. Eclipse RCP is a core component of Eclipse, which allows running component-based applications on a variety of different operating systems. Moreover, the Eclipse Foundation promotes the growth of the Eclipse Ecosystem which allows benefiting from the results of a large community of developers. Eclipse provides a well supported and stable environment to build component-based applications. Another reason to choose Eclipse was the fact that the framework is open source. So the source code is available and enabled us to change the framework where necessary.

In a first version of BSCWeasel, we basically implemented the main features of the web-based BSCW client [cf.: 1]. Later on, we added components, called plugins in the Eclipse terminology, to realize new functionality. A set of new plugins offer tools for synchronous cooperation based on the XMPP/Jabber instant message protocol. We also developed a plugin which allows the fat client to deal with more than one BSCW server. Additionally, we extended the awareness functionality of BSCW and implemented a caching mechanism.

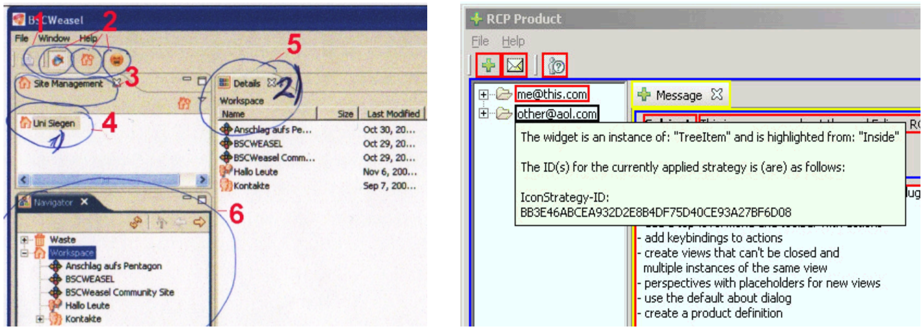
## 5 Collaboration among Users

To support collaborative appropriation activities among users, we suggest making help functions highly context sensitive and to augment help functions by functionalities of a community system. In our work, we draw on Wikis to augment help functions. Wikis are widely spread and allow editing texts in a collaborative manner.

We decided to represent the traditional help text of each function within a Wiki. Users can extend, change or annotate these texts. They can create different local descriptions of purpose, usage, or outcome of a function and exchange knowledge concerning the appropriation of this function within their local practices. Access to the Wiki needs to be highly contextualized at the user interface to select those Wiki entries which are associated with the current usage. In our approach, we took the state of the application as a proxy for the actual context of use. By means of the Meta Object Protocol and runtime reflection [18], we linked Wiki/help pages technically to specific states of the application.

From a user's perspective, a Wiki page refers to a function perceived by the users at the interface of the application, and therefore, supports appropriation discourses among communities of users (also addressing diversifying sub-communities). The user first selects the object in question and then presses F1 to open the corresponding help/wiki page. So, the software application offers a built-in communication channel among users and therefore acts as a boundary object for contextualizing the discussion among users (see section 3).

The Wiki discourse infrastructure was realized using standardized software interfaces, but the realization of context sensitivity is more challenging. We used context identifiers in the applications source code to anchor wiki widgets in a certain functionality area. However, this implementation strategy turned out to be hard to maintain since designers may either forget to write help texts for an identifier or place the context identifier at the wrong position in the source code.



The widget is an instance of: "TreeItem" and is highlighted from: "Inside"

The ID(s) for the currently applied strategy is (are) as follows:

IconStrategy-ID:
BB3E46ABCEA932D2E8B4DF75D40CE93A27BF6D08

- add keybindings to actions
- create views that can't be closed and multiple instances of the same view
- perspectives with placeholders for new views
- use the default about dialog
- create a product definition

**Fig. 2.** Highlighting the point of interest: (left) from a user perception (right) from a computational perceptive (the tool tip refers to information that can be gathered by algorithmic reflection on current state of the application)

However, the situatedness of work activities ([37]) is a tough challenge for the underlying assumption that the execution position in the code is an appropriate measure for the current work context. Still manual maintenance of context identifiers would be quite error prone, as well. Therefore, we studied in which way users make sense of the "set of pixels generated and managed by a computational process that is the result of the computer interpretation of a program P." [5]. In our empirical studies of users' perception we present the users several screenshots of known and unknown programmes and ask them to highlight their point of interest (cf. Figure 2 left). In these studies, we observe that the way users give the pixel a meaning is related to the widget hierarchy of the interface. Based on this observation we created an algorithm which identifies function compounds as they are perceived by the users and maps them with stable context identifies. The calculation of the stable context identifier use the runtime reflection feature [18] to gather information that allows a computational identification of the point of interest (the tool tip in Figure 2 shows some of the information that was available for that widget via runtime reflection) [cf. 14].

The identified widget was highlighted as a potential point of interest at the interface (cf. Figure 2) and using the calculated context identifier as a shared reference point it offers access to the corresponding public Wiki page (cf. Figure 4) .

To implement the communication channels among users as described above, we have developed the CHIC-architecture (Community Help in Context) [36]. CHIC consists of three generic software modules: Application Integration Module (AIM), Context-Aware Adaptation Module (CAM), and Community-based Help System (CBHS) (see Figure 3).

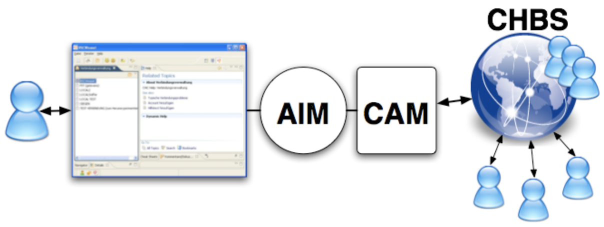
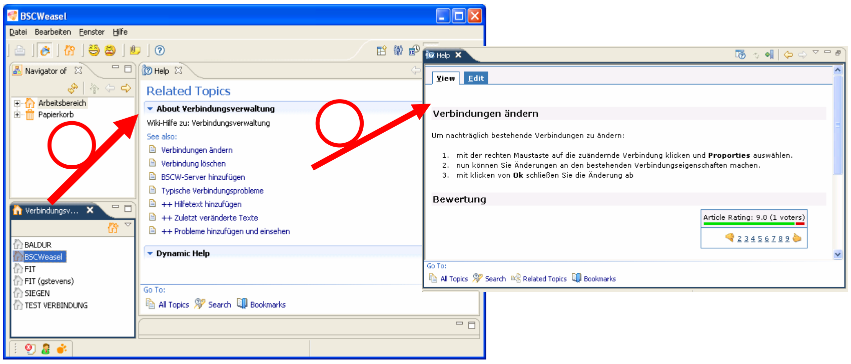


**Fig. 3.** Architecture of Community Help in Context (ChiC)

The Application Integration Module (AIM) integrates CHiC into an existing application and the user interacts with CHiC using it. When the user asks for help by pressing F1, it highlights the user perceived functions mapped to a context identifier and offers a “single-click” access to the CBHS-System [45] (for the interface see Figure 2). In order to provide this functionality, AIM requests the necessary information from the Context-Aware Adaptation Module (CAM). CAM mainly calculates the context identifier and mediates between AIM and CBHS. The CBHS can be any community system, like a Wiki, which provides an infrastructure for help discourses.



Wiki-Hilfe zu: Verbindungsverwaltung

Verbindungen ändern
Verbindung löschen
BSCW-Server hinzufügen
Typische Verbindungsprobleme
++ Hilfetext hinzufügen
++ Zuletzt veränderte Texte
++ Probleme hinzufügen und einsehen

Verbindungen ändern

Um nachträglich bestehende Verbindungen zu ändern:

1. mit der rechten Maustaste auf die zuändernde Verbindung klicken und **Proporties** auswählen.
2. nun können Sie Änderungen an den bestehenden Verbindungseigenschaften machen.
3. mit klicken von **Ok** schließen Sie die Änderung ab

Bewertung

Article Rating: 9.0 (1 voters)

2 3 4 5 6 7 8 9

**Fig. 4.** Changing the selected interface element triggers a recalculation of the help entries (1). A click on one of the help entries opens the Wiki page via the internal web browser (2).

In the BSCWeasel case, we use the Eclipse framework to integrate the Wiki help into the application context. We benefit from the Eclipse architecture which allows adding new help items dynamically. A help item implements the interface IHelpResource which delivers the subject labels of help texts and the URLs of the corresponding Wiki pages. The subject labels of help items are displayed as links in the help window of Eclipse. When a user clicks on the label, Eclipse opens the internal web browser and loads the associated web page (cf. Figure 4). To realize CAM under Eclipse, we extended the IContextProvider. IContextProvider is invoked whenever the state of the application has changed. CAM uses this trigger to inspect the actual system state and requests CBHS to return a set of help entries.

The CBHS module was realized by integrating the Atlassian Confluence Wiki[1] because it provides a commenting function, several notification mechanisms like mail, RSS, and the recently changed pages. Moreover, it provides a well defined Web Service API.

## 6 Collaboration between Users and Developers

To offer collaboration support for users and designers, we have integrated a professional requirements tracking system into the BSCWeasel application and have equipped it with a specific interface for the users.

With regard to designers' needs, our goal was to minimize the overhead from the administration of direct user feedback together with other sources of requirements. To encourage contributions from a wide variety of different users, we wanted to provide a gentle slope of increasingly more complex levels of participation [22] in the requirements specification process. Legitimate peripheral participation in the requirements specification process is supported by allowing end users to just mark shortcomings in their current interface. However, lead users can use the system to discuss and test newly designed features in interaction with the professional designers who can use the system also for their work (e.g. design planning and scheduling).

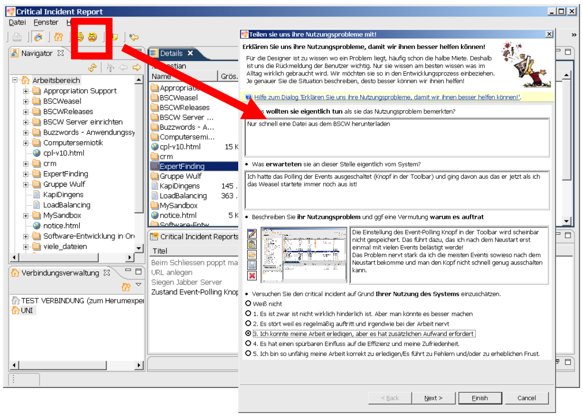
To realize this part of the appropriation infrastructure, we came up with a hybrid approach which combines an external requirements tracking system with an Eclipse plugin which is integrated into the BSCWeasel user interface. The plugin provides specific views on the requirements tracking system. Technologically we drew on the Web Service API/remote method invocation interface of the requirements tracking system to integrate its user interface into the BSCWeasel application.

We decide to use a professional requirements tracking system, called JIRA. JIRA is a web based application supporting the interaction among developers. JIRA allows saving requirements in textual form, which can be annotated with attachments, e.g. log files or screenshots. Users of JIRA can discuss these requirements, prioritize and vote for them. A configurable workflow allows processing these requirements within the team of developers. The functionality of JIRA can be used via a web-based interface or it can be integrated into 3rd party products via the Web Service API.

The integration into BSCWeasel was realized implementing an Eclipse plugin called PaDU (Participatory Design in Use). PaDU packages JIRA's Web Service API and makes it available for Eclipse RCP applications. If a requirement is submitted to JIRA or information is retrieved from JIRA, PaDU will carry it out via the XML RPC. To lower the barriers for users, PaDU uses the integrated web browser of Eclipse. When the user wants to see detailed information about his contribution, PaDU will open the corresponding web page.

[1] http://www.atlassian.com/software/confluence/

PaDU allows contributing to the design process directly from the BSCWeasel user interface. PaDU integrates two buttons into the user interface of the BSCWeasel application (see Figure 5). The buttons help distinguishing between critical incidents (a subjective breakdown of tool usage) and use innovations (a new way of using existing functionality or a new idea for interesting functionality). These buttons are always visible and they are used as access points to document problems or suggest new design ideas.



**Erklären Sie uns ihre Nutzungsprobleme, damit wir ihnen besser helfen können!**

Für die Designer ist zu wissen wo ein Problem liegt, häufig schon die halbe Miete. Deshalb ist uns die Rückmeldung der Benutzer wichtig. Nur sie wissen am besten wissen was im Alltag wirklich gebraucht wird. Wir möchten sie so in den Entwicklungsprozess einbeziehen. Je genauer Sie die Situation beschreiben, desto besser können wir ihnen helfen!

Hilfe zum Dialog 'Erklären Sie uns ihre Nutzungsprobleme, damit wir ihnen besser helfen können!'.

s **wollten sie eigentlich tun** als sie das Nutzungsproblem bemerkten?

Nur schnell eine Datei aus dem BSCW herunterladen

- Was **erwarteten** sie an dieser Stelle eigentlich vom System?

Ich hatte das Polling der Events ausgeschaltet (Knopf in der Toolbar) und ging davon aus das er jetzt als ich das Weasel startete immer noch aus ist!

- Beschreiben Sie **ihr Nutzungsproblem** und ggf eine Vermutung **warum es auftrat**

Die Einstellung des Event-Polling Knopf in der Toolbar wird scheinbar nicht gespeichert. Das führt dazu, das ich nach dem Neustart erst einmal mit vielen Events belästigt werde!
Das Problem nervt stark da ich die meisten Events sowieso nach dem Neustart bekomme und man den Kopf nicht schnell genug ausschalten kann.

- Versuchen Sie den critical incident auf Grund **Ihrer Nutzung des Systems** einzuschätzen.

○ Weiß nicht
○ 1. Es ist zwar ist nicht wirklich hinderlich ist. Aber man könnte es besser machen
○ 2. Es stört weil es regelmäßig auftritt und irgendwie bei der Arbeit nervt
[✓ 3. Ich konnte meine Arbeit erledigen, aber es hat zusätzlichen Aufwand erfordert]
○ 4. Es hat einen spürbaren Einfluss auf die Effizienz und meine Zufriedenheit.
○ 5. Ich bin so unfähig meine Arbeit korrekt zu erledigen/Es führt zu Fehlern und/oder zu erheblichen Frust.

**Fig. 5.** PaDU's access point is in the button bar which activates the requirements tracking system

When a user presses one of these buttons, a multi-page dialog window appears. The dialog is adapted from the critical incident dialog [4] by Hartson et al [8]. Beyond purely textual descriptions of the requirements, we integrated features which allow for ostensive and deictic references to the software artefact in order to clarify design ideas. We have, for instance, extended the dialog window to enable users to add screenshots, annotate them textually or graphically, and attach own sketches. PaDU automatically takes a snapshot of the current state of the BSCWeasel interface at the moment it is activated. A drawing tool is available to edit the screenshots.

Designers can deal with the contributions of the users in the same way they do with any other requirements documented in the system. They can discuss these requirements, prioritize them and vote for them. To offer accountability with regard to their inputs, users can see all activities that happen in the requirements tracking system. Via their interface, users can track the state of their contributions. They are informed via email in case someone comments on their input. They can also set up links to other entries in order to be informed about the state of their procedure. Additionally, designers can send a direct email to a user to clarify open issues.

However, the discourse culture which emerged in the BSCWeasel project was slightly different. Instead of writing an email, questions to a contributor were attached as a comment. The contributor received an email containing this comment and had the option to answer to the email by adding a new comment. As a result, a public discourse around certain requirement emerged.

We understand design to be a communicative process which needs to be transparent to those who want to participate.[2] In order to satisfy this requirement users and designers should have similar rights with regard to inspecting the requirements database and adding comments. To support users in becoming familiar with the web interface and to increase their awareness of the design process, PaDU's start page contains all the contributions made by this particular user.

Additionally, we save a user's contributions locally. So, writing a design suggestion can happen before it is published within the requirement tracking system. Users can see all of their ideas in a list. A double click on published design ideas opens the web browser and shows the corresponding web page in the requirement tracking system. The web page shows the contribution in detail, the state of the contribution in the overall design process, and discussions and comments added in reaction to the contribution.

## 7 Bridging between Product-Oriented and Process-Oriented Flexibility

With regard to product-oriented flexibility, the current BSCWeasel implementation is grounded in the features which Eclipse RCP provides. A plugin is in a technical sense *the smallest application unit of the Eclipse Platform function that can be developed and delivered separately [16].* Such a component must be designed according to the Eclipse plugin mode which is an extension of the common OSGi standard. Roughly spoken a component is a bundle of java code, additional resources, and a description of the component's properties.

Product-oriented flexibility is basically limited to extensibility. The Eclipse Update Manager allows high-level components to be integrated at runtime into a composition to provide additional functionality. Plugins for an application are stored in specific web sites and have to follow the update site's specification. From this site they can be downloaded to the local plugin directory.

Compositions of plugins cannot be reassembled during runtime by end users since Eclipse RCP does not provide any specific user interface for that. Contrary to FreEvolve [37], Eclipse does not connect the component structure with the corresponding elements at the user interface.

Beyond extensibility, Eclipse RCP implements an interface-related aspect of product-oriented flexibility which is part of the Eclipse workbench concept. The user interface of an application is subdivided into different areas in which different interface elements (called views) can be placed. These areas can be recursively split when needed. Users can reposition these interface elements to compose a new integrated user interface and enhance the functionality by adding new views.

[2] This aspect distinguish our approach, e.g. from the concept of remote evaluation promoted by Hartson et al. (1996). In their work end users should only deliver information of shortcomings in the design. However, their participation in the design-related discussions of these shortcomings is not technically supported.

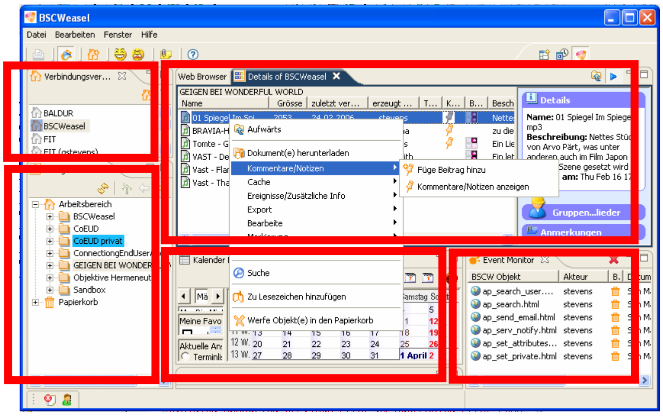


**Fig. 6.** Screen shot of an Eclipse workbench with a set of BSCWeasel related views (outlined with a rectangle)

Figure 6 provides a screenshot of the interface of the BSCWeasel, which illustrates the Eclipse workbench concept. Typically at the beginning the BSCWeasel user interface displays only some views and with the time the user interfaces become more complex, presenting more sophisticated features (like the Event Monitor in Figure 6, which presents awareness information).

We have set up an agile software development process to be able to react immediately to user requirements expressed within PaDU. To bridge between product- and process oriented flexibility, the developers can build new plugins or modify existing ones by means of short release cycles. We practice refactoring, as a method for architectural evolution. Eclipse as a software development environment offers tool suites to support these approaches to process-oriented flexibility, like refactoring feature, Release Engineering support, etc.

## 8 Case Study

Both prototypes which we described here were implemented based on the Eclipse plugin framework. Together with further applications that are being used at the periphery of other implementations (e.g. email clients), they form an infrastructure to support appropriation work in the late phase of usage design. Any application which operates on the same infrastructural background (Eclipse) would be able to use our concepts.

To evaluate our concepts, we implemented them into the BSCWeasel client. BSCWeasel was developed by a research group of a German university. The core group consisted of two developers which were complemented temporarily by different student teams. Contrary to most work in the area of product-oriented flexibility, we applied an agile development process which was directed towards short release cycles and an immediate evaluation in practice.

In May 2005 an initial version of BSCWeasel was used by the developers and their student team. Later versions were announced towards the research group at the university (about 15 members) and towards two groups at a research institute in applied computer science (about 15 researchers) 100 km away from the university. All researchers were basically familiar with BSCW, though the system was applied to rather different degrees. The appropriation process of BSCWeasel was analyzed via the discussion threads provided by PaDU and CHIC. Moreover, observations and informal interviews were carried out to explore the appropriation of BSCWeasel further on. Additionally, two studies were conducted based on the ISO 9241-10/12 standards to improve the usability of the application. The first study was carried out in April 2005 with nine users. It focused on the basic functionality of BSCWeasel. In January 2006, a second study with six users looked particularly into the usability of the CHiC and PaDU functionality.

With regard to the appropriation of BSCWeasel at the university and the research institute, we know about 10 regular users. They were intense users of BSCW before and identified specific BSCWeasel functions to be incorporated into their practice. The individual "killer" functions were not part of the BSCW thin client and covered a wide range of functionality. Some of them were requested via PaDU – like the option to download more than one file or complete folder structures, or a synchronized view on local and remote directory structures. Other functions were communicated directly towards the team of developers.

About half of the BSCWeasel users have made use of PaDU. From September 2005 to July 2007 130 design requirements were expressed via PaDU. Due to the relatively small number of active users the design team was rather reactive towards their suggestions. About 50% of these proposals got implemented.

In evaluating our experiences, we will focus on two main issues. First, we will investigate into the impact the appropriation infrastructure had on the design process. Secondly, we will look into the relations and interferences among the different functions of the appropriation infrastructure.

## 8.1 Grounding Design in Practice

After the roll-out of PaDU, the designers got more feedback from users. Since PaDU items were stored in the Bug Tracking System, the feedback was more systematic and easier to handle and became an integral part of the coordination work carried out by the designers.

PaDU is mainly used by users to make designers aware of a usability problem and/or feature request, however discussions among designers and users happened rarely. This may be due to the fact that PaDU does not disclose the users' identity. However, we found frequent instances in which contributions made in PaDU triggered a reflection process within the design team, e.g. discussing design alternatives related to a concrete user experience. Sometimes designers react to a user comment, when requirements expressed by the users were not clear (e.g. a designer wrote: "Well, technically this is a little thing [to implement the feature request]. However, for the moment is not yet clear to me how you would like to use it") or different solutions were possible, (e.g. asking which of different options to implement an "open file with …" feature would be needed).

Most of the contributions made by the users referred to cases in which they were able to accomplish their task, often by means of a workaround, but wanted a better support from BSCWeasel. The snapshot annotation tool was typically used to point to the referred area in the user interface. The suggested redesign would render more control or efficiency to their work. For example, with regard to the upload function a user made the following proposal: "It would be a nice thing to know the data volume ahead of an upload. In this case one would know how long it takes and whether there is sufficient space available".

Analyzing the contributions made via PaDU, we found little design requirements which went far beyond the given functionality. Most of the suggestions were rooted in practical experiences using BSCWeasel in the users' daily work. Accessing PaDU directly from their context of use seems to stimulate users to focus on present-at-hand technology when contributing. It seems to result in incremental rather than highly innovative suggestions for redesign.

However, these contributions, based on practical experience, had a considerable impact on the design process. One of the developers came up with the following bon mot: "If programming is understood as theory building [24], PaDU helps making it a 'grounded theory'".

Nevertheless, PaDU should be perceived as an additional instrument to improve distributed, continuous Participatory Design and not as a replacement for traditional, creativity oriented Participatory Design instruments like Future workshops.

## 8.2 Integrating Different Functions in an Appropriation Infrastructure

When integrating the different parts of the appropriation infrastructure and studying them simultaneously, we became aware of the phenomena of interference. The lacking integration of users' communication channels with those channels between users and designers created problems. The segregation of the different appropriation support functionalities – such as help, adaptation, or requirements articulation – seems to be dysfunctional.

We observe that CHiC was mainly used as a traditional help system with only little discussions among users going on. It seems that CHiC and PaDU cannibalized each other since both could be applied when BSCWeasel was not present-at-hand. This fact became obvious in the second usability study. An interviewee stated that he is occasionally uncertain whether to address other users or better the developers. He had a problem in connecting the BSCWeasel client with the BSCW server. Reflecting on his problem, he was not sure whether it was caused by bad design or inappropriate use. So he could not decide easily whether to discuss his problem in PaDU or CHiC. In another case a user explained that she put a question into PaDU but later cancelled it. She was not sure whether this issue was just her personal problem, ("just not knowing enough about the system"), or if the issue was more generally relevant for the design of BSCWeasel. These findings seem to indicate a need for a deeper integration of PaDU and CHiC.

Another example for lacking integration is the gap between flexibilization at the level of the user interface compared to the level of the component structure of the application and its missing integration into a communication infrastructure. Eclipse's "viewer" concept offers an elegant solution for the composition of interface elements compared to user interfaces of web-based clients augmented by applets. All interface elements can be integrated into a combined view, called perspective. We observed that this feature was applied by the users to individualize their user interface. However, Eclipse still suffers from the fact that this interface layer of a user centric composition is not connected to the underlying component structure. So, the underlying structure is not visible and cannot directly explore from the user interface taking the actual use context into account. Obviously, lacking references between software structure and user interface leads to confusion and does not support users in understanding the linkage between the user interface and the software architecture [6].

As a result, users may develop a mental model which diverges strongly from the software architecture. It leads specifically to problem in cases where applications, such as Eclipse IDE or BSCWeasel, are composed by hundreds of components provided by different vendors. During our usability study we found an example for these phenomena.. It turned out that users assumed that our chat tool (a 3rd party component) and the BSCW system where tightly coupled because the interface elements were integrated. In another case we observed an Eclipse IDE user who had problems in finding out which vendor was responsible for a specific view which he had added to his user interface. He was looking for more information about the object in question.

Moreover, Eclipse suffers from lacking integration of the component management features into a community-oriented communication infrastructure. The Eclipse community starts to become aware of this problem. In particular, some commercial companies like Innoopract have started to extend Eclipse with a component repository service with thousand of plug-ins. They support end users to assemble their personal Eclipse configuration out of the repository in an easy way. Furthermore one can observe that traditional centralized provisioning strategies will be enhanced by concepts that support a grassroots diffusion of composition and tailored artifacts.

## 9 Conclusion

Support for appropriation work has to be understood as a core challenge in the field of End User Development. From the perspective of appropriation work, the concept of design needs to be re-interpreted. It should be understood as designing usages, not tools. In such a perspective, activities of end users such as configurating, tailoring, sense making, or negotiating conventions of usage have to be linked to the work performed by software developers. Appropriation and realization are dialectic moments in usage design, while in the late phase it is mainly driven by actors and stakeholders from the use sphere, not from the design sphere. These activities can be considered as inherently collaborative and should be explicitly supported by appropriate infrastructures build into the applications. Extending earlier research, we aimed for an infrastructure that does not only support user-user-collaboration, but also integrates the professional designers' work sphere. When supporting appropriation work, it seems to be necessary to go beyond traditional EUD techniques such as configurating or tailoring and connect professional designers with end users.

As a first case of an appropriation infrastructure, we developed collaborative functionalities based on the Eclipse plugin architecture. We integrated this infrastructure into BSCWeasel, a rich client for the web-based groupware system BSCW. The first functionality we provided – supporting user-user collaboration - was a context-aware extensible help system based on a wiki metaphor. Help texts could be complemented or specialized for certain situations of usage. The second functionality we provided – supporting user-designer collaboration – was a requirements tracking system that allowed for rich technology-related articulations and collaborative requirements management and was integrated into the application, as well. Together with additional tools which were integrated, they form an infrastructure for appropriation work.

We collected first evaluation data by means of an empirical case study covering the infrastructure's appropriation in three research groups. Some of the assumptions guiding our design have been confirmed by the study. First of all, there is an interest, if not even a need, for collaboration in the appropriation of technology. It makes sense to understand the application to be appropriated not only as a boundary object between design and use, but also as a communication anchor and medium for appropriation activities. Figure 1 illustrates the fact that both the appropriation infrastructure and the component-repositories mediate between the developers and user communities if such a modularized design is meaningful for the different actors.

From a user perspective, the following activities seem to be expressions of appropriation work which are grounded in the reflective use of an application:

- consulting help systems,
- making requirement inquiry and (re-)design contributions,
- tailoring and updating an application.

Therefore, appropriation activities should be conceptualized from a holistic perspective. Appropriation infrastructure should integrate the different support features to a wider extend than BSCWeasel has done so far. The infrastructure needs to be tightly integrated into the software artifact for an optimal support of usage design:

- the communication channels should be activated directly from the access point of the functions they refer to [cf.: 44],
- the communication channels should be structured according to the way the users perceive the functionality,
- the communication channels should offer opportunities to create deictic references towards specific aspects of the functionality.

In order to better support appropriation work, the linkage between product- and process-oriented aspects of flexibility implies further fundamental design challenges. Users and designers need to build common ground with regard to the component structure of an application. Users build their mental models of technology based on the *perceived* functionality. Designers work is typically grounded in a long professional tradition of software modularization which has led to a separation of application logic and user interface. However, when supporting appropriation work, this tradition needs to be challenged since it is the source of misunderstandings between designers and users.

Our research needs to be extended to a theoretical level (e.g. connecting it to the discourse around 'infrastructuring', [33]) as well as on a technological level (e.g. fine grained component based tailorability beyond plug-in integration, additional support functions, different infrastructural background technologies, e.g. service-oriented architectures). Ultimately we hope to be able to establish a methodological perspective on end user development understood as software (usage!) design which is not dominated by the traditions of programmers but respects the work of all stakeholders involved.

We conclude this paper with a refinement of the definition of EUD, picking up the consideration that EUD should support a continuous co-evolution of both, the system and the user [5, 10]. In times where software development methodology conceptions like ‘perpetual beta’ [48] becomes general accepted designers, co-workers and other stakeholders of the software artifacts are essential participants in the continuous co-evolution. This also means that personal and shared design activities as highly interwoven. A definition of EUD should be reflected this issue, thus we suggest a refinement as follows: *EUD denotes a set of methods, tools and techniques to support end users to enforce their interests in the continuous co-evolutionary process by modifying individual artifacts or participating in the modification of shared artifacts.*

## Acknowledgements

We would like to thank IBM for supporting our research by means of an Eclipse Innovation Award. We are also grateful to the German Science Foundation and the German Ministry of Research and Education (BMBF) for funding in the field of End User Development (EUD).

## References

1. Appelt, W.: What Groupware Functionality do Users Really Use? In: Proceedings of the 9th Euromicro Workshop on PDP 2001. IEEE Computer Society, Los Alamitos (2001)
2. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Pearson Education (2000)
3. Bentley, R., Dourish, P.: Medium versus mechanism: Supporting collaboration through customisation. In: Proceedings of ECSCW 1995. Kluwer Academic Publishers, Stockholm (1995)
4. Bentley, R., et al.: Supporting Collaborative Information Sharing with the World Wide Web: The BSCW Shared Workspace System. In: The World Wide Web Journal: Proceedings of the 4th International WWW Conference, vol. 1, pp. 63–74 (1995)
5. Costabile, M.F., Fogli, D., Mussio, P., Piccinno, A.: Visual Interactive Systems for End-User Development: a Model-based Design Methodology. IEEE Trans. on SMC - Part A: Systems and Humans 37(6), 1029–1046 (2007)
6. de Souza, C.S., Barbosa, S.D.J., Silva, S.R.P.: Semiotic engineering principles for evaluating end-user programming environments. Interacting with Computers 13 (2001)
7. Dourish, P.: Developing a Reflective Model of Collaborative Systems. ACM Transactions on Computer-Human Interaction 2(1), 40–63 (1995)
8. Finck, M., Gumm, D., Pape, B.: Using Groupware for Mediated Feedback. In: Proceedings of the Participation Design Conference 2004 (2004)
9. Fischer, G., Girgensohn, A.: End-user modifiability in design environments. In: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM Press, Washington (1990)
10. Fischer, G., Ostwald, J.: Seeding, Evolutionary Growth, and Reseeding: Enriching Participatory Design with Informed Participation. In: Proceedings of the Participatory Design Conference (PDC 2002). 2002. CPSR, Malmö (2002)

11. Fischer, G., Giaccardi, E.: Meta-Design: A Framework for the Future of End User Development. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End User Development, pp. 427–458. Springer, Heidelberg (2006)
12. Floyd, C., Reisin, F.-M., Schmidt, G.: STEPS to Software Development with Users Source. In: Proceedings of the 2nd European Software Engineering Conference. LNCS. Springer, London (1989)
13. Gamma, E., Wiegand, J.: The eclipse way processes that adapt (2005)
14. Grüttner, M.: Entwicklung eines generischen Visualisierungs- und Interaktionskonzepts für kontextsensitive Hilfesysteme und prototypische Implementierung für das Eclipse RCP-Framework. In: Wirtschaftsinformatik. University of Siegen: Siegen (2007)
15. Henderson, A., Kyng, M.: There's No Place Like Home: Continuing Design in Use. In: Greenbaum, J.K. (ed.) Design at Work - Cooperative Design of Computer Artifacts, Hillsdale, pp. 219–240 (1991)
16. IBM, Draft: Eclipse Platform Technical Overview, IBM Corporation and The Eclipse Foundation (2005)
17. Karsten, H., Jones, M.: The long and winding road: Collaorative IT and organisational change. In: Int. Conference on Computer Supported Work (CSCW 1998). ACM Press, New York (1998)
18. Kiczales, G., des Rivières, J., Bobrow, D.: The Art of the Meta-Object Protocol. MIT Press, Cambridge (1991)
19. Lieberman, H., Paternó, F., Wulf, V. (eds.): End User Development. Springer, Berlin (2006)
20. Lippert, M.: Eclipse Core - Unter der Haube, Teil 2: Ein Blick auf den Entwicklungsprozess des Eclipse-Plattform-Projekts. Eclipse Magazin (2006)
21. Malone, T.W., Lai, K.-Y., Fry, C.: Experiments with Oval: a radically tailorable tool for cooperative work. ACM TOIS 13(2), 177–205 (1995)
22. McLean, A., et al.: User tailorable systems: Pressing the issues with buttons. In: Proceedings of CHI 1990, Seattle, Washington (1990)
23. Mørch, A.: Three Levels of End-user Tailoring: Customization, Integration and Extension. In: Kyng, M., Henderson, H. (eds.) Computers and Design in context, pp. 51–76. MIT Press, Cambridge (1997)
24. Naur, P.: Programming as Theory Building. Microprocessing and Microprogramming 15, 253–261 (1985)
25. Ngwenyama, O.K.: Groupware, social action and organizational emergence: on the process dynamics of computer mediated distributed work. Accounting, Management and Information Technologies 8(4), 123–143 (1998)
26. Oberquelle, H.: Situationsbedingte und benutzerorientierte Anpassbarkeit von Groupware. In: Hartmann, A., et al. (eds.) Menschengerechte Groupware - Software-ergonomische Gestaltung und partizipative Umsetzung, pp. 31–50. Stuttgart, Teubner (1994)
27. Orlikowski, W.J.: Evolving with Notes: Organizational change around groupware technology. In: Ciborra, C. (ed.) Groupware & Teamwork, pp. 23–60. J. Wiley, Chichester (1996)
28. Orlikowski, W.J., Hofman, J.D.: An Improvisational Model for Change Management: The Case of Groupware Technologies. Sloan Management Review, pp. 11–21 (Winter 1997)
29. Pipek, V.: From Tailoring to Appropriation Support: Negotiating Groupware Usage. In: Faculty of Science, Department of Information Processing Science 2005. University of Oulu, Oulu (2005)
30. Pipek, V., Kahler, H.: Supporting Collaborative Tailoring. In: Lieberman, H., Paterno, F., Wulf, V. (eds.) End-User Development. Springer, Berlin (2006)

31. Pipek, V.W.: A Groupware's Life. In: Proceedings of the Sixth European Conference on Computer Supported Cooperative Work (ECSCW 1999). Kluwer, Dordrecht (1999)
32. Rumpe, B., Schröder, A.: Quantitative Untersuchung des Extreme Programming Prozesses (2001)
33. Star, S.L., Bowker, G.C.: How to infrastructure. In: Lievrouw, L.A., Livingstone, S. (eds.) Handbook of New Media - Social Shaping and Consequences of ICTs, pp. 151–162. SAGE Pub., London (2002)
34. Stevens, G.: BSCWeasel – How to make an existing Groupware System more flexible. In: Demo presentation on the 9th European Conference on Computer-Supported Cooperative Work (2005)
35. Stevens, G., Budweg, S., Pipek, V.: The BSCWeasel and Eclipse-powered Cooperative End User Development. In: Proc. Workshop Eclipse as a Vehicle for CSCW Research at the Int. Conf. on CSCW 2004, Chicago, IL, USA (2004)
36. Stevens, G., Wiedenhöfer, T.: CHIC - A pluggable solution for community help in context. In: Proc of the 4th NordiCHi (2006)
37. Stiemerling, O.: Component-Based Tailorability. In: Institut für Informatik III, Rheinische Friedrich-Wilhelms-Universität, Bonn (2000)
38. Suchman, L.: Located accountabilities in technology production. Scandinavian Journal of Information Systems 14(2), 91–105 (2002)
39. Suchman, L.A.: Plans and situated actions: the problem of human-machine communication. Cambridge University Press, Cambridge (1990)
40. Törpel, B., Pipek, V., Rittenbruch, M.: Creating Heterogeneity - Evolving Use of Groupware in a Network of Freelancers. Special Issue on Evolving Use of Groupware, Computer Supported Cooperative Work: The Journal of Collaborative Computing (JCSCW) 12(1-2) (2003)
41. Twidale, M.B.: Over the Shoulder Learning: Supporting Brief Informal Learning. Computer Supported Cooperative Work 14(6), 505–547 (2005)
42. von Hippel, E., Katz, R.: Shifting Innovation to Users via Toolkits. Management Science 48(7), 821–833 (2002)
43. Wulf, V.: Evolving Cooperation when Introducing Groupware – A Self-Organization Perspective. Cybernetics and Human Knowing 6(2), 55–75 (1999)
44. Wulf, V., Golombek, B.: Exploration environments: concept and empirical evaluation. In: Proc. of the GROUP (2001)
45. Wulf, V., Golombek, B.: Direct Activation: A Concept to Encourage Tailoring Activities. Behaviour & Information Technology 20(4), 249–263 (2001)
46. Wulf, V., Rohde, M.: Towards an Integrated Organization and Technology Development. In: ACM Proceedings of the Symposium on Designing Interactive Systems (1995)
47. Wulf, V., Pipek, V., Won, M.: Component-based tailorability: Enabling highly flexible software applications. Int. J. Hum.-Comput. Stud. 66(1), 1–22 (2008)
48. Wikipedia: Perpetual beta. Online resource (November 28, 2008), http://en.wikipedia.org/wiki/Perpetual_beta