Andersen, R., & Mørch, A. I. (2009). Mutual Development: A Case Study in Customer-Initiated Software Product Development. Second International Symposium on End User Development (EUD 2009). Lecture Notes in Computer Science. Heidelberg: Springer (forthcoming).

Mutual Development: A Case Study in Customer-Initiated Software Product Development

Renate Andersen and Anders I. Mørch

InterMedia, University of Oslo,
P.O. Box 1161 Blindern, N-0318 Oslo, Norway
renate.andersen@ementor.no, anders.morch@intermedia.uio.no

Abstract. The paper is a case study of customer-initiated software product development. We have observed and participated in system development activities in a commercial software house (company) over a period of two years. The company produces project-planning tools for the oil and gas industry, and relies on interaction with customers for further development of its products. Our main research question is how customers and professional developers engage in mutual development mediated by shared software tools (products and support systems). We have used interviews with developers and customers as our main source of data, and identified the activities (from use to development) where customers have contributed to development. We analyze our findings in terms of co-configuration, meta-design and modding in order to name and compare the various stages of development (adaptation, generalization, improvement request, specialization, and tailoring).

Keywords: customer-initiated product development, software development, case study, empirical analysis, theoretical perspectives, mutual development

1 Introduction

The goal of the research reported here is to identify areas where end-user development (EUD) and professional software development interact. We have observed and participated in development activities in a commercial software house (referred to as company in the remainder of the paper) over a period of two years. We propose a model of the activities, which we refer to as mutual development. The model consists of the 5 sub-processes, which connects EUD and professional development.

1.1 The Case

The company is engaged in commercial software development in the area of project planning and management and provides consultancy services in using its tools. At present, the company employs 25-30 people, but they intend to grow and is

concurrently expanding their staff and searching for new markets. The main market has been the Nordic oil and gas industry. To expand into new markets, particularly building and construction, the company has started to modify and improve its knowledge management practices regarding customer relations. As researchers, we were invited by the company to give advice for how to improve knowledge management practices with customers.

The company is known for their customer initiated product development approach, i.e. close interaction with customers to develop tailor-made products [1][31]. Customers are encouraged to report problems, innovative use, and local development to the company. This has been stimulated through long-term relationships (maintenance contracts) and user forums. Each year the company hosts a large showcase where customers are invited, and developers provide communication and information sharing tools for customer interaction. This started with the telephone, then supplemented by mail, later extending to a Helpdesk interface, then a Customer Relationship Management (CRM) system, and most recently a Web 2.0 prototype created by the research team [29].

Despite their small size, the company is recognized as a major player in the business of project planning tools. They have several hundred customers and they have long-term commitments with many of them. One of their recent products is an add-on to Microsoft Project.

Our main research question and objective is how there is mutual development between customers, professional developers mediated by software products and ICT support systems in the company we studied. By *mutual development* we mean that both professional developers and end users contribute to development as active participants in both design and use. We identify the range of end-user development activities (from use to design) taking place in the interaction between the company's developers and some of their customers.

We have identified five sub-processes (adaptation, generalization, improvement requests, specialization, and tailoring) by pinpointing what developers and customers are doing and where their activities meet and overlap. We base our analysis on interviews with developers, consultants, and customers, and on data from a video-recorded workshop. The findings are compared with previous research in EUD and analyzed in terms of co-configuration [7][8], meta-design [10][12] and modding [15][16]. The goal is to identify the interdependencies of EUD and professional development and to construct a model for their mutual development.

The rest of the paper is organized as follows. It starts with an overview of EUD. Next, we present a survey of research in the intersection of EUD and software development. Then we present three theoretical perspectives on EUD. We analyze our findings by comparing with the three perspectives. At the end open issues for further research is suggested.

2 End-User Development

End-user development is an umbrella term for research and development in end-user tools for application development. This originated with research that dealt with

technological and organizational issues of an emerging field, such as end-user programming in spreadsheets and tailorable systems [22]. Most recently, web application development has introduced a new line of R&D that shares many similarities with EUD (e.g., mashups, Yahoo pipes). However, EUD was perhaps first established as a research field with its own agenda in the European EUD-Net project (2002-3), which defines EUD as "a set of methods, activities, techniques, and tools that allow people who are non-professional software developers, at some point to create or modify a software artifact" [21]. The different approaches to EUD vary with respect to how they emphasize methods, activities, techniques, and tools, and whether they focus on creation or modification of software artifacts. Furthermore, what a software artifact means also varies among researchers. Software tools, source code, design diagrams, application units, and application development environments have been mentioned. As an example, end-user tailoring is about methods, activities, techniques, and tools for adaptation and further development of existing software applications based on direct activation of tailoring tools from the applications' user interface [25] [39].

EUD is multidisciplinary and its rationale (the *"why"* of EUD) has multiple dimensions: human-computer interaction (HCI), software engineering, and organizational use. From a *human-computer interaction* perspective, EUD is about leveraging the deployment of easy-to-use ICT and turning them into easy-to-further-develop systems [21][28][40]. From a *software engineering* perspective, EUD is supportive of the trend of producing *generic applications* [2][24]. By "generic" is meant multifunctional, domain independent, or application generators, i.e. "over designed" functionality that can be configured to different user needs [26], or domain independent tools like groupware and basic drawing functionality, or "under designed" environments that support users in creating new applications [12]. For example, a groupware system can provide different users with different access rights to shared objects [33]. From the perspective of *organizational use*, the rationale for EUD is associated with the user diversity found in organizations employing advanced ICT. Users have different cultural, educational, training, and employment backgrounds. They are novice and experienced computer users (e.g. super user), ranging from the young to the mature, and they have many different abilities and disabilities [3][23][26].

2.1 Integrated EUD

EUD interrelates with software development in multiple ways, but (to the best of our knowledge) there are few studies that have examined EUD in terms of boundary crossing of two types of organizations (developer and customers). We survey the related work below. We also include work that is not commonly associated with EUD in the survey.

Stevens and Wulf [33] presented a case study of inter-organizational cooperation from the steel industry in Germany. They analyzed the relationship between two engineering offices and a steel mill to identify patterns of cooperation that can serve as requirements for new designs. They found that there was tight coupling across organizational boundaries, but also competition between the units. EUD was proposed

in terms of a component-based framework for tailoring a groupware application at runtime. The focus was on flexible access control for sharing material stored in electronic repositories among the interacting units. The new access mechanisms could be decomposed and integrated and the users were able to realize new access mechanisms that did not already exist in the groupware. By decomposing application components into simpler ones and assembling the parts into new compounds (intermediate building blocks) and applications, users can modify existing applications and create new ones, without accessing the underlying program code [40].

Eriksson and Dittrich [9] identified the reasons why tailoring should be integrated with software development. In a case study of a Swedish telecom provider, they found it was possible to provide end-user developers with the means to tailor not only individual applications, but also the infrastructure in which applications are integrated. According to the authors, this is an area that might change faster than applications, especially in rapidly changing business contexts. To support this form of tailoring in the organization, they studied tailoring needs to coordinate better with software development activities. In another study, Dittrich and Vaucouleur [4] found that customization practices of an ERP system they studied at several sites were at odds with software engineering practices, resulting in a discrepancy in terms of integrated environments for end-user development.

In a case study in an accounting company in Norway, the activities of end user developers were followed and analyzed using Activity Theory [26]. The authors show how the organization successfully initiated a program to train super users [17] in conjunction with introducing a new software application, Visma Business (VB). The research was formulated to address how super users engage in EUD activities in order to achieve an efficient use of VB, and how EUD activities were organized. In terms of organization, there was a certain division of labor within the community: 1) between the regular users and the super users, 2) between the super users and the application coordinator (acting as local developer), and 3) between the application coordinator and the professional developers. It was also interesting to find a new role for a local developer. This person's responsibility was primarily to perform EUD activities at a general level, to work closely with some of the more experienced super users in the offices, and to communicate with the professional developers. This person generalized the results of useful EUD activities and made local solutions available throughout the company.

Explicit and implicit channels for communication between developers and users for the purpose of end user development have been proposed in a variety of contexts, especially in the area of CSCW. For example Mørch and Mehandjiev [27] demonstrated that design rationale integrated with a tailor-enabled application could support indirect communication between developers and users and thus help end user developers to further develop their applications. Along the same lines, Stevens and Wiedenhöfer [34] developed a wiki-based help system for communication and information sharing to be integrated with standalone applications. It provides online help to a community of users and thus enhances communication between developers and users with the affordances of Web 2.0. The authors claim this form of integration creates a more seamless transition between the use context and the resolution of a problem due to the familiarity users have with Wiki-based systems [34].

3 Concepts for Analysis

We analyze our findings in terms of three theoretical perspectives on end-user development in order to account for a broad array of relevant concerns, ranging from computer science to application domains to organization of work: meta-design, modding, and co-configuration.

3.1 SER Model and Meta-Design

SER (Seeding, Evolutionary growth, Reseeding) is a process model for integrating end-user development with software engineering [11]. It is different from user-centered design in HCI (e.g., prototyping) and from software engineering (e.g., specification driven methods). It has more in common with aspects of participatory design in that the SER model describes a sociotechnical environment for tailorable applications to be used over an extended period of time. It postulates that systems that evolve over a sustained time span must continually alternate between periods of unplanned evolutions by end users (evolutionary growth), and periods of deliberate restructuring and enhancement (reseeding), involving users in collaboration with designers [11].

The SER model makes a distinction between design time and use time, which distinguishes developers' activity from users' activity. Integrating these two types of software development activities is the aim of *meta-design*: a framework to provide end users with tools that allow them to tailor and further develop professional tools in their own context [10][12]. Meta-designers use their creativity to develop sociotechnical environments in which other (less technical oriented) users can be creative in their own areas of expertise. Meta-design as viewed from a software engineering viewpoint defines flexible design spaces for end-user developers. Examples are tailoring languages, application frameworks and EUD tools integrated with applications. This means the users interested in being active contributors should be supported in exploring an application's potential for being incorporated in new activities, and evolving its functionally to support new needs [10]. To the extent this can be accomplished without end users having detailed knowledge of programming, meta-design becomes a powerful framework and perspective for EUD.

The SER model has influenced the mutual development model we present below. In particular, we elaborate on evolutionary growth and reseeding and the dynamic interaction between them in the company we studied.

3.2 Modding

Modding is when users modify products by themselves, without the direct intervention of professional developers. The term is a slang expression derived from the word modify that refers to the act of modifying a piece of software or hardware, originally conceived in the gaming industry. Modding is an alternative way of

including customers in product development processes. Modding can be seen to combine EUD and participatory design, in that it combines the inclusion of customers in both early and later stages of product development, depending on the customer's needs. By adopting this activity, modding can be seen as extending the design environment approach to EUD [12][28][40] by making it possible for customers to promote an array of ideas and needs in the early stages of product development, even before a given framework exists.

The outcomes of modding, called mods, range from minor alterations to very extensive variations of the original product [15][16]. An example of modding from the gaming industry is when hardcore players create hacks and figure out how to develop software add-ons to twist games' parameters, such as the creation of a "No Jealousy" patch, which lets characters have more than one lover without either one getting jealous [20]. What is even more interesting is how the original product serves as a platform for further modding for customers.

Modding as an alternative approach to including customers in product development processes is a noteworthy concept since it engages the customer in different stages of the product development process. Modding is based on further development of an already existing platform. However, this must not be misunderstood. It does not mean the narrowing down of product development to simply be further development of already existing products, as is often the case with tailorable applications and evolutionary application development [24]. On the contrary, it appears that already existing products may be "opened up" by end-user contributions in terms of generating new ideas for functionality, new features, and even new products. In many ways, it is the concrete (executable) applications rather than the more abstract application frameworks and tailoring languages that best serve as a platform for end-user development [24].

3.3 Co-Configuration

Engeström [7] [8] adopted the term co-configuration from Victor & Boynton [35] to enhance the theory of expansive learning in order to address a new form of work that involves user participation from customers and employees in the development of products. Co-configuration implies both a new form of work and a new way of learning. Engeström draws on the empirical findings of a broadband telecommunications firm in Finland, focusing on learning as joint creation of new knowledge and new practices by multiple stakeholders [7]. Engeström, following Victor and Boynton [35], defines co-configuration as an emerging historical type of work with the following general characteristics [7]:

- Adaptive and adaptable customer products or services, or more typically integrated product-service combinations
- A continuous relationship of mutual exchange between customers, producers, and the product-service combinations
- Continuous co-configuration and customization of the product-service-customer relationship over lengthy time periods
- Active customer involvement and input in the co-configuration work

- Multiple collaborating producers that need to operate together in networks within or between organizations
- Mutual learning from interactions between the parties involved in configuration actions.

From this description, we can understand the term co-configuration as a type of work that includes active participation from customers in developing their products. One of the characteristics of co-configuration work is the great degree of customer participation required in order for it to work. For example, when developing project planning software to fit a user organization and its work tasks, it is important to include users as participants in the process since they are the ones who know what kind of work tasks the project planning tools are supposed to support. However, not all companies will benefit by such a strategy. For example, to what degree is the company dependent on involvement from customers? What happens if some customers do not see the value of being part of such co-configuration work? To what degrees do the customers actually participate? To what degree is it reasonable to expect that customers will continue to participate over lengthy time periods? It is probably realistic to assume that in today's world of mass consumption the majority of end users will not want to design or contribute to further development of the products they use. We chose to focus on those customers who took an active part in the case we report.

4 Method

Our objective is to construct a model of mutual development between customers and professional developers as seen from a EUD perspective. The case study is designed to extend our own previous efforts by treating the interaction of two organizations (developer and customer) as the unit of analysis [26][31]. We identify the sub-processes of the product development process studied. EUD is one component in this picture, but not the only one. By presenting the whole picture we wish to provide a comprehensive view of mutual development, which we present as different stages of activity, using examples and theoretical analyses to justify our claims. We used a qualitative approach as part of a case study. In addition, we used video and audio recorders to gather data. Moreover, we used open-ended interviews, focus groups and participant observations.

4.1 Categorizing Data

This section will elaborate on how the intermediate terms used to describe mutual development emerged as a result of analysis done while screening and analyzing data. The form of analysis used is 'template analysis,' which is the process whereby "the researcher produces a list of codes (a template) representing themes identified in their textual data [19]." This is both a top-down and bottom up process. Below, we have named some terms, more precisely the different stages of mutual development, representing different themes identified in the empirical findings. After identifying

these themes, the data was analyzed with this in mind, using these themes as a template. King distinguishes three features in template analysis: *defining codes*, *hierarchical coding* and *parallel coding* [19].

Defining codes is to label a section of text with a code in order to index it as relating to a theme or issue in the data that the researcher has identified as important to his or her interpretation [19]. We had the research questions in mind the first time we went through the data, but in the second round of selecting data we categorized it accordingly. The categorization of "outer loop" and "inner loop" were used as "high-level codes," and may be connected with what King defines as *hierarchical coding*.

Hierarchical coding "is codes that are arranged hierarchically with groups of similar codes clustered together to produce more general higher order codes" [19]. The high-level codes of "inner loop" and "outer loop" roughly clustered the data into two different terrains, one about customer-initiated development activity (outer) and the other about software engineering (inner). This was done deliberately to create an overview of the data. Knowing that our area of interest was mostly on the "outer loop" product development process, the data was analyzed again for topics within this domain. It was found that within the interviews there existed some sub-processes of outer loop product development. They were identified as *Adaptation, Generalization, Tailoring, Improvement Request* and *Specialization*. Using these terms or codes as a template, the data was searched again in order to support these sub-processes with empirical evidence.

Parallel coding is when the same segment of data is classified within two (or more) different codes at the same level [19]. In one instance, the same set of data excerpts was classified within the intermediate code "outer loop" and the lower order code Specialization, which is a stage within the inner loop product development. Therefore, parallel coding was used in this context.

5 Data and Analysis

At the end of the coding we ended up with the following five sub-processes (stages) of customer-initiated product development:

- *Adaptation*: Adaptation is when a customer requests an improvement to an existing product and the company chooses to fulfill the request. It becomes an Adaptation just for this customer. Sometimes, the customer has to pay for this, sometimes not.
- *Generalization*: Generalization occurs when a new version of an existing product is released and is available to more than one customer.
- *Improvement Requests*: This is when customers request the company for extra functionality, report bugs and usability problems, and is viewed from the customers' perspectives.
- *Specialization*: Specialization is when the professional developers at the company create in-house builds. This is common in inner loop development processes where professional developers improve the products for their own internal work. This could potentially result in new features, but most often it entails refining the product, reorganizing program code, and removing bugs.
- *Tailoring* is about active end users who make adaptations on their own.

We justify these stages using the data extracts and analysis below. The two first extracts define basic issues (types of process) that resurface in the other extracts and in the analyses. The last three extracts represent four of the five stages.

5.1 Excerpt 1: Types of Improvement Request

In the first excerpt, the focus is on how a developer (informant) judges the Improvement Requests of the customer. This includes making a power decision as to what kinds of Improvement Requests to consider. The power to judge whether or not a customer Improvement Request should be accepted lies in the hands of the company's professional developers. This excerpt does not go into detail about how exactly these Improvement Requests enter the company, but it does elaborate in what way the customers ask for Improvement Requests.

Informant: *Often when they (the customers) want Improvement Requests they ask me if I can make a change (to the existing product), according to some needs they have. In addition they put it (the Improvement Request) into a list we have on the Internet. We receive a lot of Improvement Requests and some of them are actually such good ideas that we want to integrate them into our products. And there are other ideas that are really bad. There are also some ideas that are not so good (but they are doable), therefore we incorporate them if they pay for it. When doing this we make special libraries for that particular customer. Then this does not become a part of the system (the product).*

Improvement Requests turned out to be an important activity for communication with the company, requiring less technical expertise than Tailoring. Excerpt 1 is an example of how customers propose changes to the company's products without doing any local development. Excerpt 1 shows that an Improvement Request is one of the prerequisite sub-processes of Adaptation. It is when a professional developer creates a new feature for an already existing product in accordance with the customer's demands. At the end of this excerpt, the informant introduces the theme of how they get good, possible (doable) and bad ideas for further development. If an idea is labeled *good* it is accepted as is. When an idea is categorized as *possible* it means that the idea is plausible, but will not become a part of the general product. It might be accepted under contract (with payment), and turns into a local Adaptation. Finally, an idea labeled *bad* is rejected outright. Implicit in this example is the assumption that the company's employees are the ones who judge whether the Improvement Requests are *good*, *possible* or *bad* and have the freedom to make those distinctions.

As seen from a meta-design and SER perspective [11][12], Excerpt 1 may be interpreted as an example of boundary crossing, namely that submitting, receiving and handling of improvement request cross the boundary of two organizations (customer and developer). It also indicates some of the decisions that have to be made before the "evolutionary growth" of an application at a specific site can be accepted into the

"reseeding" phase by company developers. In this way, Improvement Requests can help to bridge the gap between EUD and professional development.

The data in Excerpt 1 may have some commonalities with Engeström's notion of co-configuration. Item number two in the definition of co-configuration (see Integrated EUD) is about the *mutual exchange between customers, producers and the product-service combinations* [8]. Mutual exchange can be seen in this excerpt as well, between the customers issuing requests to the company and the professional developers handling these requests. The exchange for customers is getting the development they want, while the company receives money for performing the development (or more satisfied customers).

If a request is categorized as good or possible, the next stage of Adaptation takes place. During the second stage of Adaptation terms like patch, build and version become relevant, which we discuss below.

5.2 Excerpt 2: Types of Generalization

This is part of an interview one of the researchers had with one of the developers. The informant explains the software deployment (packaging) terms *patch*, *build* and *version* as part of an elaborated answer to a question about improvement requests:

Informant: *There are three levels: we have a so-called patch, which is a quick fix to some sort of a problem. This is being sent out to the customer, which is a (solution) right there and then. After the customer installs the patch, he tests if it works and then the problem is fixed. After a while, when we have made enough patches like this, we find new errors and the customers find errors and then we make a new complete program. That is what we call a build. On top of this, we have something we call versions; they could be (called) 3.4, 3.5, 3.5.1. They have more content and much more functionality.*

Patch, build and version are the developers' responses to customers forwarding Improvement Requests in the Adaptation stage, which again can lead to Specialization and Generalization. *Patch* is understood as a quick fix to a problem. Patches are packaged extensions that fit specific versions. For example, if Word is being used to write some text and one's references in EndNote are lost each time text is converted into PDF, the company could be contacted. They will fix it and send back a so-called *patch,* which is small program (a software component) that may be installed on the computer and linked with the main program, and the problem is fixed. *Builds* result if the company has had many quick fixes, similar to the example with Word, and 2nd order problems emerge (i.e., problems connected to the compatibility of patches). Then they create a *build,* which is a compiled program. Builds are associated with Specialization. Finally, a new *version* is both an extension and a generalization. It is an extension (improvement) of a build, and a generalization when a new *version* is made available to new customers and to the existing customers when

they are due for an upgrade according to their contract. Generalization is a borderline activity between inner loop and outer loop product development.

In Excerpt 2 it is evident that to a large extent, software development at the company proceeds with the SER model, as Fischer describes [11]. Excerpt 2 has a lot in common with the example Fischer uses to explain the reseeding phase, where open source software systems take some time to evolve, aided by using local (user created) extensions and the integration of patches (*evolutionary growth*), but eventually require major reorganizing in order to incorporate the patches and extensions in a coherent fashion (*reseeding*) [11]. In the company it happened like this: First the product evolves locally as a result of *patches* created in response to customer requests, and when this becomes unwieldy the company's professional developers create a *build*. Lastly, when the modifications become too numerous or are judged to be useful (good) for other (potential) customers, the developers create a new *version* of the product. However, Fischer does not distinguish between build and version. He uses the term *reseeding* for all developer activity associated with reorganizing multiple adaptations (patched systems) into unified (seamless) versions. Due to the complexity of this activity, it is useful to distinguish the multiple sub processes (types) of reseeding and the interaction between evolutionary growth and reseeding.

5.3 Excerpt 3: Improvement Request and Adaptation

Excerpt 3 below illustrates how the Improvement Requests, as elaborated in the excerpt above, are differentiated. It also shows what is meant by Adaptation.

Question:	*So, the rationale for a given upgrade lies with a specific customer, which means that a customer can be a part of setting the standards for what other customers receive.*
Answer:	*Mm, but if what one customer suggests is far off, then we just make a local adaptation for that specific customer.*
Question:	*So, this becomes a new version for you then?*
Answer:	*What we have in addition to every menu choice is a so-called user option, it is placed in an "own" library, which can be linked, and allows us to do further product development.*

What triggered the statement above is that one of the interviewers asked how the company develops their products. In sentence number two, the informant answers that if the customer's request is "far off" they just make an Adaptation for this particular customer, as long as the customer pays for it. As mentioned above, this corresponds with an Improvement Request labeled *possible*. Excerpt 3 shows how an Improvement Request labeled *good* may become available to all customers. The informant acknowledges after some hesitation and with elaboration that the customers are to some extent "defining" what other customers receive of product upgrades. They do this by suggesting Improvement Requests and other customer-initiated activities such as Tailoring. However in most cases Improvement request that are responded to by an Adaptation, providing a custom-made product for this customer by using

patches or user options with the current released version of the product. In the last sentence in Excerpt 3, the informant explains what is meant by (local) Adaptation. It is associated with a patched system installation that can be continually adapted (further developed) by user options that are deployed in a separate package (own library). When installed in the system, it appears as a separate menu with items for the various user options.

5.4 Excerpt 4: Generalization

The above excerpt introduced the term "user option," which is a special kind of patch. The related terms user option, patch and new version will be clarified in Excerpt 4 below. The excerpt illustrates the generalization process.

Question:	*Do you have other examples of customers initiating new functionality to the product?*
Answer:	*Yes, we have done it for BuildingCompany and ABB… (two large European engineering and consultancy companies)*
Question:	*What sort of new functionality did they want?*
Answer:	*Yes, well, it is. I don't remember - it was years ago. I know that when they bought the product they had specific requirements that were originally not part of the product. But we wrote it into the contract as the functionality they wanted.*
Question:	*Ok, so it was a part of the contract?*
Answer:	*Yes, they wanted it within a specific time period. Their requirements were rather demanding regarding what they wanted us to make.*
Question:	*Was it an add-on specifically made for BuildingCompany or..*
Answer:	*No, it became a part of the product. Yes, it started as a patch, what we call a user option.*

The informant underlines that a request for new functionality eventually became part of the company's general product portfolio and was made available to all their customers. It is an example of *Generalization*. It becomes clear that in this situation the request for new functionality that BuildingCompany asked for was something specific they needed. The company wrote their demands into the contract. This excerpt reiterates a point made above, that *good* Improvement Requests would be incorporated into the next version of one of their products.

The transition from Adaptation to Generalization is evident in Excerpt 4 since it describes an activity that involves one specific product (Planner) based on interaction with specific customers (Building Company in particular). The product has developed from small local extensions (patches and user options) to a basic core (in-house) version to a new (released) version where generally useful local adaptations are incorporated into the new release. We interpret the last sentence of the excerpt to mean a step-wise integration into the product (from specific to general) along three steps. It is associated with the combination of the utterance of *"No"* and *"Yes"* that signify a contradiction and disruptive (non incremental) transition (from Adaptation

to Generalization). 1-2) Yes, it started as a special type of patch (user option), which is Adaptation, 3) no, it was only later incorporated into the product, which is Generalization. Adaptation represents the two first steps. First, the extra functionality BuildingCompany asked for is a *user option*, which means it is only available for this specific customer. Second, they want to make this available for later use, so they make a *patch* that the other customers can access upon demand, for example via the company's web pages. Third, when there is a *new version* of the product, the extra functionality (patches and user options) have been incorporated in the product and therefore made available to potentially all customers. In other words, we may say that there is a gradual development of the company's products over the years, many of which are based on local development initiatives and Improvement Requests to generalized versions and back to new initiatives for further development, as new user contexts appear.

Fischer and Ostwald's SER model [11] suggests mutual dependency of evolutionary growth and reseeding, and this is supported by the findings reported here, namely that use time activity (Improvement Requests) can trigger design-time (Generalization) activity. It is also related to SER in a more indirect way, in that Adaptation as a user-oriented design-time activity can lead to Generalization.

Jeppesen underlines how a defining characteristic of modding is how "*final mods often are freely revealed,"* meaning that no users are excluded from using the new modified version" [15]. In the same way as final mods are freely available, the Adaptations made to products based on some customers' ideas become available for all customers in the Generalization stage, when the suggestions from customers are accepted and integrated into a new version of the product, as shown in the excerpt above.

5.5 Excerpt 5: Tailoring

Excerpt 5 shows how customers locally adjust a software product by end-user programming to create their own extensions. Excerpt 5, from an interview with a customer in the building industry, shows a customer stating that he has adjusted the product himself by writing code in the domain-specific language SQL.

Question:	*Have you requested any wishes or needs for local adaptations?*
Answer:	*No, we have not got any special adaptations of the products (from the company). The reason for this is because I knew a great deal about SQL from earlier experience; therefore I managed to find a shortcut (of how to do it myself). I do not know the whole structure of the system, but it is available through ordinary documentation. There you get the whole (database) table structure and that has made it possible for me to find a shortcut through Access (a proprietary database management system) and allowed me to make some special (local) adaptations.*
Question:	*So, in reality you have made your own adaptations to the products?*
Answer:	*Yes, you may say that.*

This excerpt illustrates Tailoring, which is the sub-process that most closely resembles EUD as a standalone activity. Microsoft Office Access is used in conjunction with one of the company's project planning tools for data storage.

In the first sentence of this excerpt the customer states that the company has not adjusted the products for them. It is discovered that the reason for this is because the customer has made some adaptations to the product himself. He has *tailored* the product. This was possible for the customer because the products are well documented. In addition, because this customer was familiar with SQL, a high-level database query language, it was natural for him to fix the problem himself to suit his needs. This excerpt is an example of what we refer to as *Tailoring*. In Tailoring, the customer actually locally adapts the product without any company involvement. This might mean creating a small program to work around an inefficient solution as shown in this excerpt.

The reason the customer is able to tailor the product himself is because he is an expert project manager and is interested in learning how to work around a problem or inefficient solution when it appears. In other words, he is a super user. As an example, he describes how he can access and reorganize database tables as he sees fit and in a way that meets his organization's needs. The cost of this is his time and the skills required for programming, albeit simplified with a database query language like SQL compared to programming languages like Java. The advantage is that he will be able to see results of his ideas implemented relatively quickly as compared to the turnaround time when ideas for change are submitted to the company via improvement requests. The interviewer asks if this is a way of doing local adaptation, and he confirms that his SQL programming can be perceived as such. If Tailoring is followed with an Improvement request, tailoring might contribute to further development at the general levels, as was illustrated in the previous excerpt.

In previous work, tailoring has been viewed as evolutionary application development [24]. This view ignored the role of professional development and reseeding, and explored the design space of evolutionary growth for end-user developers. According to the mutual development perspective, this view must be updated. Based on the data reported here, tailoring is better conceived of as *evolutionary design*, in the sense that the local (customer) solution serves as a design for the general (company) solution, assuming it is accepted.

The findings reported in this section have been condensed and depicted in the mutual development model shown in Figure 1. Excerpt 1 can be seen as clarifying the informants' perception of the terms *good, possible* and *bad*. Excerpt 2 has a similar role for the terms *patch, build* and *version (user options* are further distinguished in Excerpts 3 and 4). Excerpt 3 also underlines the processes of *Improvement Request* and *Adaptation*, which are related in that one feeds into the other. Excerpt 4 exemplifies the stage of *Generalization*. It illustrates how a product becomes available to all customers. Finally, Excerpt 5 illustrates *Tailoring* by showing how a customer with some programming knowledge modified the product himself. It should be stressed that we have focused on the activities that involve end users (company customers) and multiple perspectives on developer-user interaction. We do not yet have sufficient data to illustrate the *Specialization* stage.

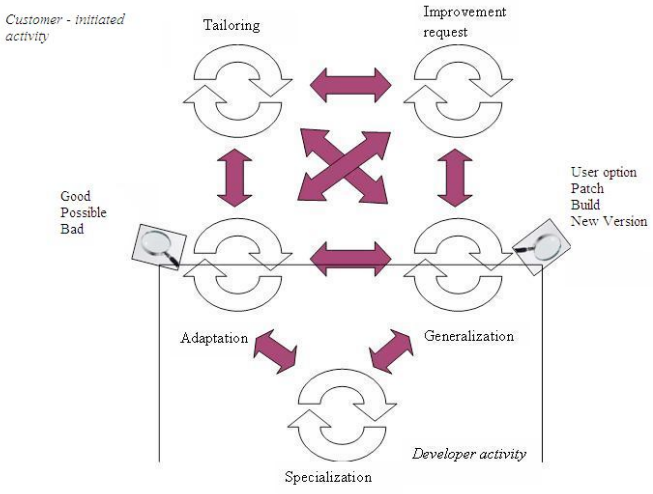

Fig. 1. Different stages of mutual development: developer activity and customer-initiated activity co-evolve; the arrows indicate dependencies. Specialization is not addressed in this paper because it does not interrelate directly with end-user activities.

6 Conclusions and Directions for Further Work

Our main research question and objective is how there is mutual development between customers, professional developers mediated by software products and ICT support systems in the company we studied. Our findings points to the components of the product development process studied. It was found that within the interviews there existed some sub-processes of mutual development (initially formulated during the preliminary analysis as customer-initiated product development) [1]. They were identified as *Adaptation, Generalization, Improvement Request, Specialization,* and *Tailoring*.

Mutual development is depicted in Figure 1. It is our first attempt to construct a model to integrate professional and end-user development [1]. Looking back, we see there are additional questions we would have liked to ask our informants, for example about the details of the customer-developer interactions. This was not possible in the current study. We cannot rule out that there may be sub-processes that have not been identified, some that may have to be modified, and yet others that need to be elaborated. This is part of future work.

In spite of this, it is clear that EUD and professional development are interdependent, and represent two different activity systems, one (customer-initiated activity) feeds into the other (developer activity) and they co-evolve. This relationship is maintained because the developer organization (company) relies on input from active customers for continuation of its products as part of maintenance and consultation contracts, and to get innovative ideas for new products that can attract

new customers. This is to some extent a result of the company's small size and its operation in a niche market. On the other hand, customers rely on the company for project planning tools, training and consultancy services, the ability to interact with the company's developers, and in general the pleasure they get from seeing their suggestions for modification being incorporated in a later version of the product.

The five excerpts we have shown to justify our claims illustrate how the products in the company have evolved from specialized and locally adapted instances to more general and stable products in interaction with customers. It goes through an elaborated process of specialization (refinement), adaptation (domain orientation) and generalization (one to many instances), starting with a stable (but non optimal) product version that is gradually extended with locally developed extensions, user options, and patches. At some point this configuration becomes unwieldy and the system is re-built. The new build may lead into a new version of the product if it will benefit the company and its other customers. Interaction between the stages is not unidirectional because new versions may lead to new local development and improvement requests, which repeat the process.

We have used theories and concepts developed by other researchers in EUD and adjoining disciplines, in particular meta-design [10][12], co-configuration [7][8], and modding [15][16] to discuss our findings at a more theoretical level. These findings are summarized as follows.

Findings According to the Meta-design and SER Perspective

- Customers being active either as designers of aspects of solutions or as producers of new ideas
- Interaction between customers and professional developers is the driving force of evolutionary development
- Professional developers adapting the products in accordance with customers' needs as main method to further develop the products
- Project planning tools evolving as a result of being used in specific contexts

Findings According to the Co-configuration Perspective

- Both customers and professional developers gain from customer-initiated product development
- Customers forwarding Improvement Requests and the company handling these form a sort of network
- Customers are active in the product development process
- Customer-initiated product development is a continuous process lasting for a long time
- When customers and professional developers interact in intimate ways to develop products, they can be considered collaborators

Findings According to the Modding Perspective

- Changes made to the company's products by users vary in complexity
- There are changes made solely by users
- Some modifications become available to all customers.
- Customer-initiated product development motivates technical-minded users

- Customers suggesting or designing new features of a product in a way "open it up" for further development
- When customers develop new features, it can be seen as a decentralized development activity

6.1 Directions for Further Work

Our results can furthermore be extended along directions advocated by researchers in user-driven innovation, participatory design, and evolution of technology.

Users can be creative and contribute to development without designing, and end-user development is often triggered by innovative use of a tool as a first step to address a breakdown in use. Norman [30] suggests workarounds and hacks as two techniques people draw on in everyday situations when coping with difficult-to-use tools. Many companies are starting to realize that innovation can arise not only from the IT department, but also from the interaction with partners, suppliers, and customers. Eric von Hippel, a pioneer and long-time champion of studying users as innovators in product development coined the term user-driven innovation. He has introduced a method for identifying sources of innovation, following "lead users" [38]. Many of the innovations he has studied originated with lead users' novel use of an existing product or an adaptation of a product based on knowledge of a related product. For example the motocross series of bikes manufactured for teenagers during the 1960s and 1970s originated as result of teenagers' desire for their bikes to resemble adult motocross bikes.

Researchers in information systems have used terms like super users [17], gurus [14], and boundary spanners [36] for a similar role as lead user. They share the view that these users help to democratize the design process, and study them by drawing on insights derived from empirical data gathered from user organizations, like we have done in this paper.

In the area of software development, participatory design [6][18], directed observation [30], and strategic ethnography [32] are methods for addressing similar issues. Directed observation means to seek out and analyze the workarounds, hacks, and clever improvisations lead users and ordinary people create at work and at home [30]. Strategic ethnography is longitudinal studies following artifacts (packaged software) as they evolve over time, which has been referred to as capturing the biography of these artifacts [32].

Based on a study of user driven innovation in an open source community von Hippel [37] observed "the ability of user communities to develop and sustain exceedingly complex products without any manufacturer involvement is remarkable." He identifies the conditions that favor user innovation and explores how circumstances evolve, sometimes to include commercial manufacturers and sometimes not [37]. When commercial manufactures are included in the loop, the resulting inter-organization activity structure can be compared with "mutual development." When commercial manufactures are not included in the loop, the resulting organization can be compared with the emerging "user manufacturing" model. Aided by the Internet and Web 2.0 applications to support communication and information sharing and most recently "mashing" (combining existing web 2.0

applications to create new ones), this model has the potential to attract new interest in end-user development due to the enormous success of this platform to attract self-motivated contributors [13]. To leverage this potential for end-user tailoring and evolutionary design is an area for further research in EUD.

In their study, Douthwaite and colleagues [5] state the following "as technology and system complexity increase so does the need for interaction between the originating R&D team and the key stakeholders (those who will directly benefit and be penalized from the innovation)." This is a hypothesis that requires further testing. It implies when software products increase in complexity, the interaction between developers and customers must proportionally increase in order to successfully manage further development and sustain the product. Otherwise, users will seek out other products that are simpler to use. The reason for increasing customer interaction as complexity unfolds is that a successful technology represents a synthesis of the developers and key stakeholder knowledge sets, and creating this synthesis requires more iteration and negotiation as complexity increases [5]. This is a hypothesis that ought to be explored in software evolution as well, in particular when end-users are enabled by EUD environments and rich feedback channels to more experienced developers.

Acknowledgements

The authors thank Annett Hillestad who was the co-supervisor to the first author. The members of the KIKK project at InterMedia, University of Oslo: Shazia Mushtaq, Damir Nedic, Kathrine Nygård, and Espen Olsen contributed to the ideas presented here. Sten Ludvigsen and Anne Moen gave us constructive comments throughout the project. The project is part of KP-Lab (Knowledge Practices Laboratory), and supported financially by the European Commission's contract FP6-2004-IST-4 027490.

References

1. Andersen, R.: Customer-initiated product development: A case study of adaptation and co-configuration, Master's thesis, Dept. of Informatics, University of Oslo, Norway (2008)
2. Bansler, J.P., Havn, E.: Information systems development with generic systems. In: Walter R.J. Baets (eds.) Proceedings from Second European Conference on Information Systems, pp. 30--31. Nijenrode University Press, Breukelen (1994)
3. Costabile, M.F., Foglia, D., Fresta, G., Mussio, P., Piccinno, A.: Software environments for end-user development and tailoring. PsychNology Journal 2(1), 99--122 (2004)
4. Dittrich, Y., Vaucouleur, S.: Practices around customization of standard systems. In: Proceedings of the 2008 international Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '08), pp. 37--40. ACM Press, New York (2008)
5. Douthwaite, B., Keatinge, J.D.H., Park, J.R.: Why promising technologies fail: The neglected role of user innovation during adoption. Research Policy, 30(5), 819--836 (2001)

6. Ehn, P., Kyng, M.: Cardboard computers: Mocking-it-up or hands-on the future. In: Greenbaum, J., Kyng, M. (eds.) Design at Work: Cooperative Design of Computer Systems, pp. 169--195. Lawrence Erlbaum, Hillsdale (1991)
7. Engeström, Y.: New forms of learning in co-configuration work. The Journal of Workplace Learning 16, 11--21 (2004)
8. Engeström, Y.: Enriching the Theory of Expansive Learning: Lessons From Journeys Toward Coconfiguration. Mind, culture and activity 14 (1-2), 23--29 (2007)
9. Eriksson, J., Dittrich, Y.: Combining tailoring and evolutionary software development for rapidly changing business systems. Journal of Organizational and End-User Computing 19(2), 47--64 (2007)
10. Fischer, G., Giaccardi, E. Ye, Y., Sutcliffe, A.G., Mehandjiev, N.: Meta-design: A manifesto for end-user development. Comm. ACM, 47(9), 33--37 (2004)
11. Fischer, G., Ostwald, J.: Seeding, evolutionary growth, and reseeding: Enriching participatory design with informed participation. In: Proceedings of the Participatory Design Conference (PDC'02), pp. 135--143. ACM Press, New York (2002)
12. Fischer, G., Scharff, E.: Meta-design: Design for designers. In: Proceedings 3rd International Conference on Designing Interactive Systems (DIS 2000), New York, pp. 396--405 (2000)
13. Floyd, I. R., Jones, M. C., Rathi, D., Twidale, M. B.: Web mash-ups and patchwork prototyping: User-driven technological innovation with Web 2.0 and open source software. In: Proceedings of the 40th annual Hawaii international Conference on System Sciences, pp. 86--96 (2007)
14. Gantt, M., Nardi, B.: Gardeners and gurus: Patterns of cooperation among CAD users. In: proceedings of the Conference on Computer-Human Interaction (CHI '92), pp. 107--117. ACM Press: New York (1992)
15. Jeppesen, L.B.: Profiting from innovative user communities: How firms organize the production of user modifications in the computer industry. Working Papers 2003-2004, Dept. of Industrial Economics and Strategy, Copenhagen Business School, Denmark (2004)
16. Jeppesen, L.B., Molin, M.J.: Consumers as co-developers: Learning and innovation outside the firm. Working Papers, 2003-01, Dept. of Industrial Economics and Strategy, Copenhagen Business School, Denmark (2003)
17. Kaasbøll, J., Øgrim, L.: Super-users: Hackers, management hostages or working class heroes? A Study of user influence on redesign in distributed organizations. In: Proceedings of the 17th Information Systems Research Seminar in Scandinavia (IRIS-17), pp. 784--798. Dept. of Information Processing Science, University of Oulu, Finland (1994)
18. Kanstrup, A.M., Christiansen, E.: Selecting and evoking innovators: Combining democracy and creativity. In: Proceedings of the 4th Nordic Conference on HCI (NordiCHI 2006), pp. 321--330. ACM Press, New York (2006).
19. King, N.: Template analysis. In: Symon, G., Cassell, C. (eds.) Qualitative methods and analysis in organizational research: A practical guide, pp. 118--134. Sage, London (1994)
20. Knight, W.: Supernatural powers become contagious in PC game, http://www.newscientist.com/article.ns?id=dn6857 (April 28, 2008).
21. Lieberman, H, Paterno, F., Wulf, V., (eds.) End-user development: Empowering people to flexibly employ advanced information and communication technology. Kluwer, Dordrecht (2006)
22. Mehandjiev, N., Bottaci, L., (eds.) End-user development: Special issue of the Journal of End User Computing, 10 (2) (1998)
23. Mehandjiev, N., Sutcliffe, A. G., Lee, D., (2005): Organisational views of end-user development. In: Lieberman, H., Paterno, F., Wulf, V. (eds.) End user development: Empowering people to flexibly employ advanced information and communication technology. Dordrecht: Kluwer Academic Publishers.

24. Mørch, A.: Evolving a generic application into a domain-oriented design environment. Scandinavian Journal of Information Systems 8 (2), 63--90 (1996)
25. Mørch, A.: Three levels of end-user tailoring: Customization, integration, and extension. In: Kyng, M., Mathiassen, L. (eds.) Computers and Design in Context, pp. 51--76. MIT Press, Cambridge, MA (1997)
26. Mørch, A.I., Hansen Åsand, H.R., Ludvigsen, S.R.: The Organization of End User Development in an Accounting Company. In: Clarke, S. (ed.) End User Computing Challenges and Technologies: Emerging Tools and Applications, pp. 102--123. Information Science Reference, Hershey, PA (2007)
27. Mørch, A.I., Mehandjiev, N.D.: Tailoring as collaboration: The mediating role of multiple representations and application units. Computer Supported Cooperative Work 9(1), 75—100 (2000)
28. Mørch, A.I., Stevens, G., Won, M., Klann, M., Dittrich, Y., Wulf, V.: Component-based technologies for end-user development. Comm. ACM 47(9), 59--62 (2004)
29. Nedic, D., Olsen, E.A.: Customizing an open source web portal framework in a business context: Integrating participatory design with an agile approach. Master's thesis, Dept. of Informatics, University of Oslo, Norway (2007)
30. Norman, D. A.: Workarounds and hacks: The leading edge of innovation. Interactions 15(4), 47--48 (2008)
31. Nygård, K.A., Mørch, A.I.: The Role of Boundary Crossing for Knowledge Advancement in Product Development. In: Proceedings Int'l Conf. Computers in Education (ICCE 2007), , pp.183--186. IOS Press, Amsterdam (2007)
32. Pollock, N., Williams, R.: The biography of the enterprise-wide system or how SAP conquered the World. Routledge, London (2008)
33. Stevens, G. & Wulf, V.: A new dimension in access control: Studying maintenance engineering across organizational boundaries. In: Proceedings of CSCW'92, 196-205. New York: ACM Press (2002)
34. Stevens, G., Wiedenhofer, T.: CHIC - A pluggable solution for community help in context. In: Proceedings of the 4th Nordic Conference on HCI (NordiCHI 2006), pp. 212--221. ACM Press, New York (2006).
35. Victor, B., Boynton, A.C.: Invented here: Maximizing your organization's internal growth and profitability. Harvard Business School Press, Boston (1998)
36. Volkoff, O., Strong, D. M., Elmes, M.B.: Between a Rock and a Hard Place: Boundary Spanners in an ERP Implementation. In: Proceedings of the 8th Americas Conference on Information Systems, pp. 958--962 (2002)
37. von Hippel, E.: Innovation by User Communities: Learning From Open-Source Software. MIT Sloan Management review 42(4), 82--86 (2001)
38. von Hippel, E.: Democratizing Innovation. MIT Press, Cambridge, MA (2005)
39. Wulf, V., Golombek, B.: Direct activation: A concept to encourage tailoring activities. Behaviour & Information Tech. 20(4), 249--263 (2001)
40. Wulf, V., Pipek, V., Won, M.: Component-based tailorability: Enabling highly flexible software applications. Int. J. Hum.-Comput. Stud. 66(1), 1--22 (2008)