Information and Software Technology 56 (2014) 1436–1456

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Software engineering beyond the project – Sustaining software ecosystems

Yvonne Dittrich *

IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark

ARTICLE INFO

Article history:
Received 25 June 2013
Received in revised form 15 February 2014
Accepted 18 February 2014
Available online 27 March 2014

Keywords:
Software ecosystems
Software product development
Qualitative empirical research

ABSTRACT

Context: The main part of software engineering methods, tools and technologies has developed around projects as the central organisational form of software development. A project organisation depends on clear bounds regarding scope, participants, development effort and lead-time. What happens when these conditions are not given? The article claims that this is the case for software product specific ecosystems. As software is increasingly developed, adopted and deployed in the form of customisable and configurable products, software engineering as a discipline needs to take on the challenge to support software ecosystems.
Objective: The article provides a holistic understanding of the observed and reported practices as a starting point to device specific support for the development in software ecosystems.
Method: A qualitative interview study was designed based on previous long-term ethnographical inspired research.
Results: The analysis results in a set of common features of product development and evolution despite differences in size, kind of software and business models. Design is distributed and needs to be coordinated across heterogeneous design constituencies that, together with the software, build a product specific socio-technical ecosystem. The technical design has to support the deference of part of the development not only to 3rd-party developers but also to local designers tailoring the software in the use organisation. The technical interfaces that separate the work of different design constituencies are contested and need to be maintained permanently. Development takes place as cycles within cycles – overlaying development cycles with different rhythms to accommodate different evolution drivers.
Conclusion: The reported practices challenge some of the very core assumptions of traditional software engineering, but makes perfect sense, considering that the frame of reference for product development is not a project but continuous innovation across the respective ecosystem. The article provides a number of concrete points for further research.

1. Introduction

The notion of ecosystems has been used in the last years to describe software developed, maintained and evolved through collaboration of the software product developer and 3rd party developers. In [10], Bosch proposes the notion of software ecosystems for trans-organisational collaboration where 3rd party developers add functionality to a software product based on an open platform. Many of the articles written thereafter focus on how to move from a close development based on a software product-line approach to an ecosystem based development approach. (See e.g. [21] as well as the majority of the articles of the 2012 Special Issue of the Journal of Systems and Software [11].) Other research focuses on the interaction between different actors in the ecosystem: Hanssen [26] describes the journey of a software product from its enthusiastic beginnings to a mature ecosystem involving the customers in the development and opening the architecture to allow for 3rd party development. Also Janssen et al.'s research focuses on the interaction between different actors in the software ecosystem [35,36]. The role of interfaces and licenses for the interaction and architecture is explored e.g. by Scacchi and Alspaugh [54].

This article takes addresses a different question: How do software development processes change when developing software in and for a software ecosystem? The answers presented here are based on empirical research of four different software product developing companies. As Bosch in [10] remarks, many software

* Tel.:+45 7218 5177.
E-mail address: ydi@itu.dk

http://dx.doi.org/10.1016/j.infsof.2014.02.012
0950-5849/

ecosystems have existed for a long time. Successful software products are developed and evolved over several decades. The mere survival implies that the organisations developing them have been successful in keeping the development in synch with the developments of the use context supporting not only 3rd party developers but also End User Developers. However, as the research below shows, such achievement is not an easy one, and it is apparent that many of the software engineering methods, tools and techniques do not support the development and evolution of software products.

The analysis proposes that the problem might be due to the focus on the project by software engineering methods and techniques. Software engineering as a discipline has developed around projects as the central organisational form of software development. The main part of the SWEBOK chapter on Software Engineering Management addresses management of software *projects*, starting with the activities defining the scope of a software project and initiating this project [32]. The CMMI defines a project as '... a managed set of interrelated resources that delivers one or more products to a customer or end user. It has a beginning and an end and typically operates according to a plan' [12]. The definitions provided in the textbox on this side are retrieved through the Software Engineering Vocabulary from ISO, IEEE, and PMI standards [33].

Definitions retrieved from SEVOCAB [33].

project.

(1) endeavor with defined start and finish dates undertaken to create a product or service in accordance with specified resources and requirements *(ISO/IEC 15288:2008 Systems and software engineering–System life cycle processes, 4.20) (ISO/IEC 15939:2007 Systems and software engineering–Measurement process, 3.34)*.

(2) an undertaking with pre-specified objectives, magnitude and duration. *(ISO/IEC 2382-20:1990 Information technology–Vocabulary–Part 20: System development, 20.07.01)*.

(3) a temporary endeavor undertaken to create a unique product, service, or result. *(A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Fourth Edition)*.

(4) a collection of work tasks with a time frame and a work product to be delivered *(ISO/IEC 20926:2003 Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting practices manual)*.

(5) set of activities for developing a new product or enhancing an existing product *(ISO/IEC 26514:2008 Systems and software engineering–requirements for designers and developers of user documentation, 4.38) Note:* A project may be viewed as a unique process comprising coordinated and controlled activities and may be composed of activities from the Project Processes and Technical Processes.

In other words, a project provides a closure for the software development, defining borders regarding scope, and then based on the scope: effort and time. Control over scope, effort (translated into expenses) and lead time – at first glance – seem to be in the interest of both the development organisation and the organisation in and for which the software is to be used.

For many contexts in which software is developed, the assumptions behind the project organisation for software development seem to hold. However, already in 1980, the experience with the evolution of the IBM 360 operating system let Lehman to formulate the laws of software evolution [42]. Based on the understanding that the application of software is changing the application domain and with it the problem that the design is based on, Lehman declared that all software that is used experiences pressure for change. He identified a reflexive feedback cycle between usage and the design of software that is responsible for this dynamics. The current research on software evolution (see e.g. [45,46]) closely relates software evolution and maintenance, and focuses on models of software evolution based on, for example, measurements of source code changes, control and handling of feedback, or it emphasises directing and implementing software evolution.

The research below indicates that with respect to customisable and end-user configurable software products challenge the assumptions proposed by the project organisation of software development further: The delivered product is rather a half product that has to be configured and customised to a specific context.[1] Development takes place through interlacing heterogeneous design and use activities distributed across different design constituencies, assemblies of individual and organisational actors mandating a specific set of design and development decisions [64]. There is no decided beginning of the evolution, and the goal is to keep the product useful and attractive to its users over a long time, rather than delivering it once. Continuous contact with users and customers is crucial. As the use contexts change, the evolution of software products to keep in synch with the application domain developments requires continuous effort. The maintenance and communication of design and architectural knowledge take place in an informal manner. The development takes place in interlaced cycles of versions, service packs and long-term efforts that address the improvement of the overall design.

The article is based on an interview series complementing previous, long-term research collaboration with a number of companies: In a research project on software development for governmental administration, a small company developing booking systems mainly for sports places was pointed out to us by municipal partners as one providing good service. The development practice of the five-person company could only be described as agile [27–29]. Already in this research project, the difficulty of accommodating requirements with different time horizons in the same development process became visible. In a second research project on interaction design for user interface frameworks for high-end mobile phones, the dependency of the local development and the application development by the mobile phone companies was identified as the main hinder for implementing specific interaction design methods [53]. When addressing the upgrade-problem for ERP systems, the ecosystem around ERP systems [20] showed to be one of the major constraints for the technical design of customisation mechanisms [56]. A similar dependency between technical (in this case architectural) design, business and use contexts, and the organisation and practices of the software development was the result of an investigation into the re-engineering of hydraulic simulation software [59,60]. Despite the heterogeneity of the software products with regard to size, application domain, and technical sophistication commonalities between the different practices became visible.

This series of observations of development practices – all different but all deviating from recommended processes, methods and technical design – provided the motivation and basis for the interview study presented here. The article analyses the organisation and practices of four different software ecosystems around

[1] In this article, 'configuration' describes the adaptation of software through predefined interfaces. 'Customisation' is used for the adaptation of software by changing the source code using a programming language. 'Tailoring', here, covers both terms.

software products that are configured and customised for a specific context of use. The qualitative interviews are designed to complement and triangulate previous fieldwork with the companies.

The research presented here connects topics normally separated by (sub-) disciplinary borderlines. Software engineering, software architecture, HCI, Computer Supported Cooperative Work, and Information System Studies all started to address aspects relevant for software ecosystem evolution, failing in their partiality to grasp the essential interaction between the dimensions. The article design, therefore, does not follow the traditional pattern but pulls in related research in the discussion where relevant. To provide a background for the method section and the analysis, the following section introduces the researched software product ecosystems. Section 3 thereafter discusses the research methods.

Section 4 presents the analysis of the interviews along three themes:

First, design and development are distributed across heterogeneous design constituencies. The development needs to be understood as an interlacing of mutually dependent design and development activities. An important issue is the cooperation across the respective ecosystems and especially with the end-user in order to communicate user innovations and developments in the use-context to inform the development.

The second theme concerns the technical design. If the software has to be adapted to different use contexts, part of the design is deferred as customisation or configuration to the organisational implementation and even to the use context. One function of the design is the separation and coordination of the development of the different design constituencies. The architecture and design tends to become the tacit knowledge of a community of practice rather than an explicit document evolving with and coordinating the development.

The third theme addresses the development practices. Product evolution needs to juggle the emerging requirements that are a result of the dynamics of the heterogeneous design constituencies in the ecosystem as well as maintain an evolvable design. The evolution process as it unfolds is a result of an overlay of different cycles in order to accommodate the dynamics of the use context, the evolution of the implementation base, as well as the need of long-term maintenance and improvement of the architectural design. Balancing different qualities and sets or requirements – new features to improve the quality in use; product related qualities like architectural structure, performance, defects; and process qualities such as lead-time – is a major challenge and is addressed by different ways of organising the development process.

The discussion section relates the analysis to previous research and develops the contribution, which is summarised and reflected upon in the conclusion.

The main contribution of this paper is to better understand how the development and evolution of software products differs from project-based development regarding the interaction of the different actors, the technical design, and, as a result of the former, the organisation of the development. The notion of product specific ecosystems allows understanding the observed and reported practices as rational rather than deficient. This provides the base for future development of methods, techniques and tools to support an increasingly important way of software development.

2. The four software product ecosystems

This section introduces the four companies. The comprehensive presentations serve two purposes: On the one hand, the different products and history are introduced, and on the other hand, the specific ecosystems are presented. Both provide a background for the presentation of the analysis results along specific themes. As the ecosystems, on one hand, are part of the analysis result, but, on the other hand, are necessary to follow the discussion of the research methods, they are presented in an independent section before the method discussion in the next section.

2.1. UIQ – developing a user interface framework for high-end mobile phones

UIQ develops a user interface framework plus an application suite for high-end mobile devices that include a phone. The interface platform and the application suite are based on the Symbian operating system. UIQ also develops specific applications for their customers, the producers of mobile phones. These producers, e.g., Motorola and Sony Ericsson, which owned UIQ, develop their own applications based on the UIQ platform.

Another group of customers are software developers that develop independent software using the UIQ platform based on a licensing scheme. The telephone companies develop several series of mobile phones based on the same version of the UIQ platform. Mobile phones are seldom updated, once they are sold. If so – e.g., by downloading a new version from Sony Ericsson's home page – the whole technology stack is exchanged.

UIQ is not directly selling to users of mobile phones, and few phone users are aware that they are using UIQ's software. Nonetheless, usability is an important quality. The interaction design team members therefore have built up a network of pilot users whom they consult for usability tests and long term usability studies. Fig. 1 presents UIQ's ecosystem.

The two interviewees had different roles in the organisation. One interviewee comes from interaction design and works as an

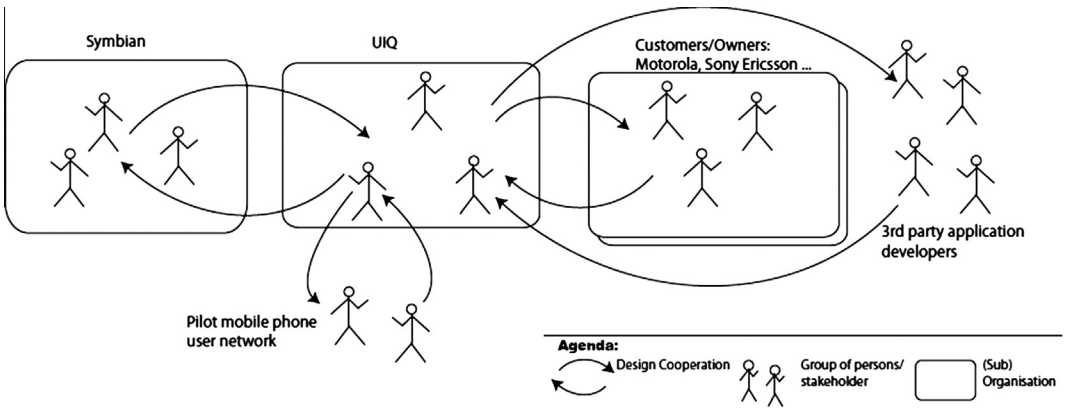

Fig. 1. UIQ's ecosystem.

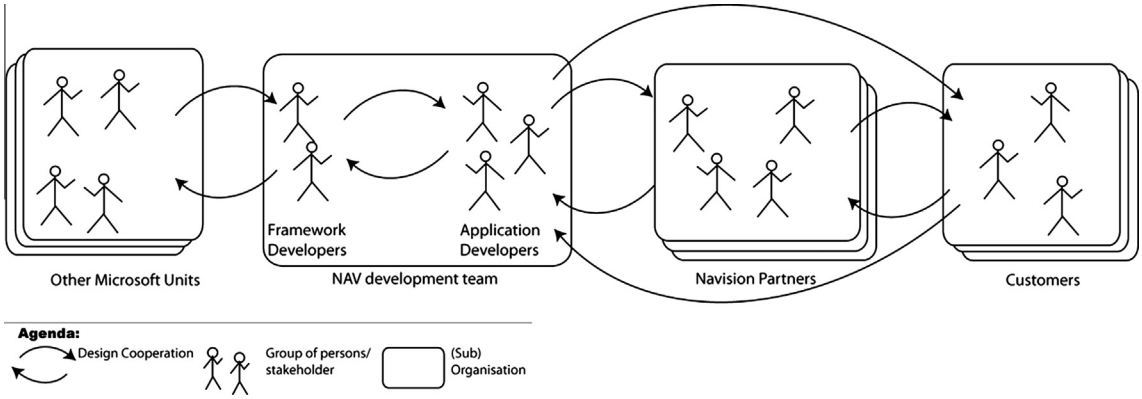

Fig. 2. Microsoft Dynamics Nav's ecosystem.

interaction architect, which means that he is a member of a group that has the responsibility to promote an integrated development of platform and application suite. The second works as a quality manager with the development of roles, responsibilities, and processes. He has been employed about one year prior to the interview and, due to his experience with another product development organisation, has been trusted with a reorganisation further discussed in the analysis.

From 2000 until 2005, as senior researcher and supervisor, this author took part in empirical research at the company. The involvement is documented in [52,53,18]. Through supervisory meetings, participation in joint workshops, project meetings, and supervision of design, implementation and analysis of empirical research, the author developed a thorough knowledge of the company and its organisational development and practices.

Shortly after the interview, Nokia took over the Symbian and UIQ shares from its co-owners and made both products open source. This resulted in the collapse of both companies.

2.2. Microsoft Dynamics – evolving an ERP system for small and medium sized companies

The Microsoft Dynamics development organisation is part of the Microsoft concern developing business solutions that use Enterprise Resource Planning (ERP) systems for different market segments. Dynamics Nav is one of these ERP systems. There are about 200 people involved in the development. At the time of the interview, a proprietary base technology was being replaced with a Microsoft compatible base.

The base software is adapted to the legislation in different countries, and is configured and customised to the specific organisation by partners, that is consultancies specialising in implementation and adaptation of Microsoft ERP systems. To do so, the partners have access to a development environment where they can adjust any part of the application. The term framework is used to describe the programming language and modelling base. Both are used to implement the application, the runtime system and the development environment. Fig. 2 shows Dynamics Nav's ecosystem. It differs from a similar one published [20], as it here focuses on the aspects of the ecosystem that became apparent in the interviews.

The interviewee worked as a release manager for 3 different Microsoft business products. The role of the release manager roughly corresponds to a project manager in project based software development.

From 2003 until today (2010), the author was also involved as supervisor and key investigator in a cooperative project 'Design of Evolvable Software Products' with the development centre in Copenhagen. This involvement is documented in one PhD thesis [62], a technical report and an article [19,20] and two Master theses [15,2]. Similar to the cooperation with UIQ, the author took part in joint workshops, project management meetings, as well as in joint empirical research.

2.3. SIM – Water and Environment Simulation

Water and Environment Simulation[2] (SIM) is a Danish not for-profit organisation whose headquarters is situated in a university related science park outside of Copenhagen. It started as a spin-off of a hydraulics department of the Technical University of Denmark. SIM offers consultancy and software services related to water and environmental issues. The consultancy is partly based on hydraulic simulation software that is also sold as software products.

Traditionally the software has been used in consultancy projects resulting in a report based on simulation runs. Through a graphical user interface, a hydraulic system is defined. In the case of one-dimensional simulations, the system consists of a network describing the waterways along with various other parameters describing the system, e.g., vents and locks influencing the flow of the water in specific points. The system definition, along with temporally varying data about inflow into the system, is used as input data to the so-called simulation engines. The simulation engines are solvers of differential equations describing the physics of the system. The results of the system are typically time series of water level and discharge at various locations in the system. Originally, SIM's business was centred around consultancy and system development based on the software products. Hydraulic engineers and water management experts use the software to produce studies evaluating the result of changes to a water system. This can be the river dams for energy production, the effect of changes to a city's water supply or sewer system, or the stratification of rivers. One of the more high profile projects was the evaluation of the effect of the Øresund bridge and tunnel between Denmark and Sweden on the water exchange between the Baltic and the North Sea.

Over time, the software has been used more and more to support control systems for e.g., water supply and sewage systems, so called 'on-line' systems. Here the simulation runs continuously, partly based on real time measurements from the water system. Results of the simulation are not only used to monitor the water system but also directly or indirectly – via an operator – to control the water system. Here the software and a number of tools are used as a set of frameworks to develop a custom monitor and control system. This change in usage leads to a different set of requirements, e.g., different simulation engines might have to be

[2] Name changed for confidentiality reasons.

integrated. The company took part in a European project developing a standard for interfacing simulation engines.

When new modelling elements are needed, the software is customised, that is, the source code is changed. The customisation is done by the hydraulic engineers who have basic programming skills. Although the software is mainly used by internal consultants, it is also sold as a product to both consultancies and end-user organisations.

At the time of the interviews SIM was undergoing an organisational change. Previously the company was organised according to consultancy domains, e.g., Urban water systems, Water resources, and Marine Simulation. The software products were maintained and evolved in these departments. The hydraulic engineers applying the software were also developing the software. Some of the engineers focused more on the software development. The organisation separated the software product development organisationally as SIM products from the hydraulic simulation projects in SIM solutions. Fig. 3 presents the new ecosystem of SIM's hydraulic simulation products. The change in organisation and especially the reflection of its rationale highlights the challenges the product development provided and the trade-offs the organisation had to handle.

Three persons were interviewed: the head of the new software products unit, a senior consultant who had worked as a head of product for one of the one-dimensional simulation engines, and a developer mainly working with the new decision support systems.

The author was principle investigator in a cooperative research project with SIM 'Design of Evolvable Software Products' around software architecture and evolvability 2006–2009. The author was involved in managing, planning and implementing the empirical research. Later, a second PhD project has conducted empirical fieldwork on the use of Instant Messaging in distributed development together with the company. The involvement is documented in [59,17].

2.4. UIB – software distribution for local area networks

In addition to developing customer specific GIS based application, the Umwelt Informatik Büro (Environmental Informatics Office, UIB) supports, maintains and evolves an open source product (OPSI http://www.opsi.org/) steering the distribution of software in a local area network. It is situated in Mainz, one of the German regional capitals. The first version of the software was developed for a regional environmental agency between 1995 and 1998. As the software was built upon other open source products, the decision to run it as an open source product 'was nearly legally necessary.' The company lives today from offering courses and support contracts for the product and customer specific contract development that feed into the open source line; it also uses the product itself doing system administration for governmental agencies; the users who are part of the open source community share procedures handling background installations of specific programs.

About 20 governmental agencies and smaller companies are paying customers, that is, have signed a support contract and, in some cases, sponsor development projects of different sizes. At the time of the interview, another 40 system administrators download the software and the user documentation and handle the installation and use themselves. The software fills out a market niche, as competitive off-the-shelf products are too expensive for small companies. The German government has decided on a policy to use open source products where possible.

At the time of the interview, two and a half persons work on the maintenance and evolution of the software; two developers and the head of development and managing director of the company are the ones mainly responsible for major developments and support. About four to five persons work with customer specific system administration, often sitting directly at the customer's site. Three persons work with customer specific GIS applications, and two mainly work with administration and accountancy. As the company has an active policy to support flexible working times, this does not necessarily mean that the company has 12 full time employees. However, apart from the main development group, the internal users and the open source community members are invited to contribute to the development, and they do so from time to time.

The interviewee was the development lead and managing director of the company. Fig. 4 presents the OPSI ecosystem.

The author has known the managing director of the company as well as other employees privately for more than 10 years. Throughout this time, the software development practices, the open source business model, and new revisions have been discussed and compared to the author's empirical research in other companies. The tight cooperation between the two lead developers was also subject of observation, as the author witnessed the release of the revision during a visit three months before the interview. As the discussions were not expected to become informal interviews and part of empirical research they were not documented.

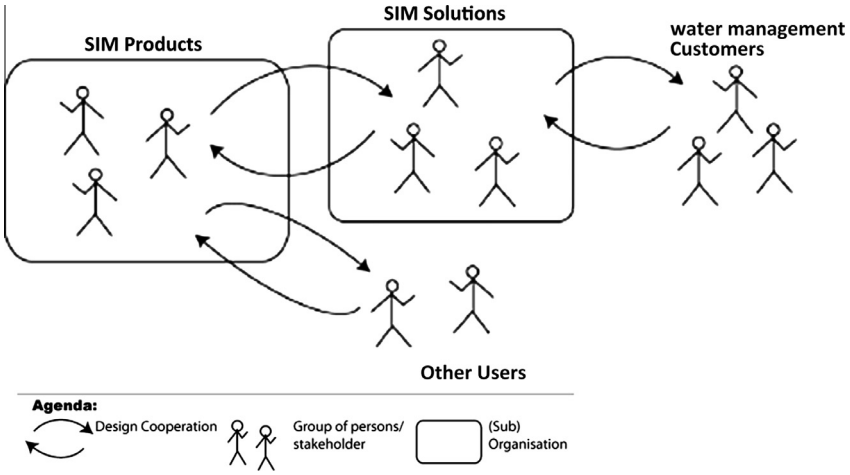

Fig. 3. SIM's ecosystems.

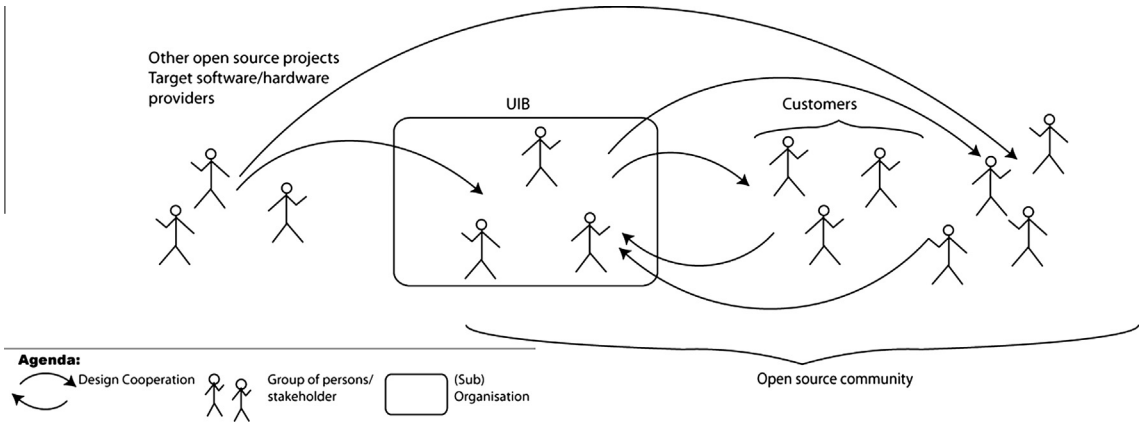

Fig. 4. Opsi's ecosystem.

3. Research method

The goal of the research is to map out the ecosystem of different stakeholders involved in the process, the organisation of the development processes and team, the influence of different quality goals and the role of the software architecture in the development. As we were aware that each of the software products was part of its own specific ecosystem, a qualitative interview study was designed [51]. In the following, Section 3.1 explains the case selection, Section 3.2 details the prior fieldwork and presents the interview design. Section 3.3 describes the analysis process, and Section 3.4 discusses the measures taken to assure the trustworthiness of the research. Section 3.5 discusses the limitations of the research.

3.1. Case selection

The motivation for the study is based in qualitative research on software development and organisational implementation and usage of software since 1999. Out of this body of research, three relevant companies were chosen. The companies were chosen because the author still had contact with company members and management so that interviews could be arranged. Given the small sample size, additional companies were interviewed in order to expand the empirical base. This turned out to be successful in only one case. As described above, the author had privately been acquainted with several members of the company for more than 10 years. The development practices were subject to discussions and unplanned observation. This informal fieldwork was not documented, as it was never meant to be used as empirical material. Two other interviews were excluded from the analysis, as follow-up questions and member checking turned out to be impossible because the interviewees had left their respective company. The information in these cases could also not be triangulated with previous fieldwork. The research approach thus is a hybrid between a multiple case study and an interview study. To compensate for this shortcoming, substantial effort was made to assure the trustworthiness of the findings.

3.2. Data collection

As mentioned in the introduction, the research presented here is based on prior fieldwork with three companies, in which the author participated as the supervisor, and a long-term contact with several members of the fourth company. The time frame and character of this fieldwork is detailed in Table 1. The prior formal fieldwork addressed different research questions. It was thus neither possible nor meaningful to go back and perform a full analysis of

Table 1
Overview of interviewees and prior fieldwork in relation to the cases.

Company	Interviewees	Prior fieldwork	Related publications
UIQ	1 Interview with 2 interviewees: Interaction Architect Quality Manager	5+ years of fieldwork with the company focussing on communication between interaction designers and software engineers. This fieldwork included participatory observation, extended interviews and action research	[52,53]
Microsoft	One interview with a Release Manager ('project manager' for the releases)	Research on customisation and configuration practices with consultancies and Independent Software Vendors Research cooperation and supervision on design of configuration mechanism Supervision of research on the introduction of business processes to support development, tailoring, and use of the software product Supervision of a M.Sc. thesis on coordination of distributed development This fieldwork included interviews of other actors in the ecosystem, analysis of video and audio recordings of customisation, and document analysis	[19,20,2,15]
SIM	3 Interviews with 3 interviewees: Head of software product department Senior consultant simulation system Developer decision support systems	Research focussing on software architecture, 2005–2008 Previous research included participatory observation, source code analysis, document analysis and participatory action research	[59–61]
UIB	1 Interview with the Managing director and development lead	Private contact with company for more than 10 years Recurring discussion on development process and development practices with the interviewee as well as other members of the company Informal observation of coordination and communication around a major release	

the prior collected field data. Prior research was used to develop the interview questions and to formulate follow-up questions during the interview, which served as a base to understand the answers. In the analysis, the prior research was also used to triangulate the findings.

An interview was developed with the purpose to explore the following topics: (1) the rhythm and rationale of the development process, different quality goals, and how they influence the development, (2) contact and relationship to customers and users, (3) organisation of the development, and (4) the role of software architecture in the development. The interviews took place based on an interview guideline on the premises of the companies. The guideline was a very rough one, serving as a reminder for the interviewer to discuss certain issues. It was adjusted to the company and the role or the interview partner. In Appendix A, the interview guideline for the third Interview with SIM can be found. Reference to prior empirical research with the company allowed to pose follow-up questions and to discuss the advantages and disadvantages for recent organisational changes. The interviewees were decided upon by the contact persons of the companies. In some instances, the contact persons themselves were among the interviewees. On average, the interviews took one hour, after which they were transcribed, summarised, and member checked with the interviewees. This mail exchange was used in two cases to pose follow-up questions.

3.3. Analysis

The interviews were transcribed and coded using grounded theory-inspired coding practices [58]: in an open coding cycle, the themes and categories were developed. This was done by reading the interview transcripts and applying keywords identifying the contents of each utterance. As the set of codes developed, a second coding phase, axial coding, related the contents of the different interviews and, at the same time, cluster the codes in categories and subcategories. Here, a common coding scheme is used, which in this case was developed by developing summaries of each interview. The third analysis step, selective coding, relates the codes and the respective interview parts so that the central themes and their relationship become visible and can be integrated into a theory. As indicated above, we did not develop a grounded theory but only employed the coding strategies. The result of the coding can be found in the structure and contents of Section 4. To complement the interview material, the member checking round was used to gather additional information.

For two reasons the analysis cannot be called a grounded theory approach: First, the research is based on a combination of long-term formal and informal case study and interviews. The understanding of the topics, the jargon used and the relation, e.g., to the history of the organisational development and changes to it, are based on the previous formal and informal research. Two additional interviews with companies who were not involved in previous or parallel research projects were discarded, as it was not possible to triangulate the research or to do in-depth follow-up interviews.

The previous fieldwork, however, was not subject to the grounded theory inspired analysis: It either addressed other research topics or was on a very informal basis. In the analysis, we explicitly refer to the earlier published research when this additional information is used. Member checking was applied both on an interview level and with regard to the whole analysis.

Second, as the empirical base for the article is a combination of long-term case study research and directed interviews, it was not possible to extend the sample until saturation of the categories was reached. A trial to include additional cases was discarded due to the reasons detailed above.

3.4. Trustworthiness

To assure trustworthiness [51]: we had the interviewees check the interview summaries and the company specific parts of the analysis; we used the contact to pose follow-up questions where necessary, and we triangulated the interviews with knowledge from previous research in three of the case organisations. The comparison of 4 cases presents an additional level of triangulation. Rich descriptions are provided in the analysis, thus allowing the reader to critique our conclusions. Below these measures are described in greater detail.

3.4.1. Rich descriptions

The analysis provides descriptions including citations from the interviews. This is complemented with information from prior research that is published in other places. These rich descriptions are not to be misread as unscientific story telling; rather, they provide the reader with the possibility to more easily follow the analysis and to critique the conclusions.

3.4.2. Member checking and follow-up questions

Apart from triangulation, member checking is one of the corner stones of quality assurance of qualitative research. The analysis is distributed to the research subjects or partners, and their feedback provides an indication of the quality of the analysis. Member checking was used on two levels. Both a summary of the interviews as well as the respective analysis parts were distributed to the interviewees in order to check whether our interpretation was in line with the interviewees' understanding. The feedback and confirmation confirm the analysis.

Qualitative research has the characteristic that from the beginning the researchers do not necessarily know what the relevant questions are. The research itself is used to develop the conceptual frame. It is therefore crucial to be able to return to the interviewees to complement the field material. In our case, this was done together with the first round of member checking.

Two interviews were excluded from the analysis, as the member checking was not possible due to interviewees changing employment. Accordingly, these interviews were excluded from the analysis.

3.4.3. Triangulation

The interview data was triangulated with prior extended empirical research in the same companies. The previous research has consisted of participatory observations, formal and informal interviews, document and code analysis. The comparison with the previous research allowed us not only to ask deepening questions, but it also provided us with a background for understanding what was referred to in the interviews. It furthermore allowed us to relate the interviews to perspectives of other members of the organisation. This way, the earlier research compensated for the low number of interviewees in each company. Moreover, the hypothesis that product development takes a different and distinctive organisational form – beyond the project – is based on this prior research.

Furthermore, the four cases triangulate our findings with respect to size of development organisation – a few developers to several hundred developers, kind of software – system administration support, user interface frameworks for mobile applications, ERP-systems, and scientific software, and business model – distribution through partner companies, in-house and external users, open source, and customer-owned.

One of the cases that we had to exclude from the analysis would have provided a negative case regarding part of the analysis. The software was a true shrink-wrap product that did not require any customisation or configuration. The results are therefore limited to configurable and customisable software products.

The discussion brings in additional research addressing product development – two interview studies and one case study – and research on usage of software products in other contexts are used to triangulate the findings of this study.

The extensive member checking that we applied, together with the ability to address missing points with follow-up questions and the case specific triangulation with prior empirical research, provide a sound base for the reliability of the representation of the cases in the analysis. The cross case triangulation and the triangulation with research not related to the case organisations support the claim of generalisability beyond the individual cases. However, as with case studies, even with multiple case studies, further research is needed to identify both the extent and the limitations of the observed phenomena. The rich descriptions provide the reader with possibility to judge the analysis, discussion and conclusions.

3.5. Limitations

The research presented here focuses on the software development in software ecosystems from the perspective of the software product developer. Though we have in other contexts implemented research with other actors of the respective ecosystem, this presents a limitation of the research presented here.

4. Analysis results

When analysing the interviews, three main themes became visible: (a) The design of the used implementation of the software is distributed over different design constituencies, sometimes located in two or more organisations. We therefore take up the analysis by addressing this topic. (b) The software architecture is an important asset but not necessarily explicitly documented. Architectural structures are maintained as shared understanding of the architects' or lead developers' community of practice. It needs to support the distribution of design over different constituencies. (c) The development continuously proceeds in cycles rather than in a project's goal-driven manner. Albeit sometimes appearing as agile, this term does not describe the overlay of differently timed development cycles driven by different rationales. The reader might benefit from referring to the Figs. 1–4 of Section 2 for illustration of aspects of the analysis presented below.

4.1. Relating design across different constituencies

In all four cases, the product provides mechanisms for their users to adapt the software to a specific application context, i.e., an organisation, a local area network to be managed, a simulation project, or a specific implementation of mobile applications. The character of these systems of interlaced design constituencies is mirrored in the technical implementation and in the organisation of the development. The complementary issue is then how to assure that the feedback from the different design constituencies informs the further development of the product. This is mentioned as one of the important challenges. Our different cases provide a set of different, but essentially similar, approaches.

4.1.1. Interlacing design constituencies

Software products aim at providing generic functionality useful in a set of different contexts. There are two possibilities to achieve this goal: either the software product can be defined so that it supports an isolatable task, with defined interfaces to other systems, or, as in our cases, the software provides possibilities for consultants or for users to adapt the product to the context in which it is then used. The results of adaptations and 'design in use' are through the ecosystem fed back into the main development organisation. The ecosystem is also used to recruit domain experts and, in some cases, to sponsor more comprehensive development activities. In the following, we describe the specific interlace of design constituencies related to the different cases.

UIQ

On the one hand, the user interface framework, UIQ, depends on and feeds requirements to the underlying operating system. As the operating system is developed by another company, UIQ here acts as client or user. On the other hand, the framework is used by customers to develop brand specific applications and interfaces. Development takes place in the form of projects where customer requested functionality and applications are developed. The framework is evolved and released together with the new features. Beyond the core group of customers, who also own the company, a user community that uses the framework for application development is supported. UIQ maintains a network of mobile phone users in order to keep track of developments in the way the devices are used in everyday life.

The different development processes of Symbian, UIQ and its customers are not always in sync which results in implementation, e.g., of functionality belonging to the framework being developed by the application developers or functionality belonging to the operating system developed by UIQ. This sometimes results in code being moved from one company to another to avoid code duplication, and placing the functionality where it belongs from an architectural point of view.

The ecosystem is also the base for recruiting users that are both willing to take part in usability studies and sponsors for development projects. The customers – who also own UIQ – go together in sponsoring development processes.

Microsoft Dynamics

Microsoft Dynamics is selling its ERP-products through a network of so-called partner companies, who configure and customise the software for the user organisation. Within the user organisation, the software is often adapted, e.g., when the organisation changes. While configuration is done through the run time interface, the customisation that changes and adds code is done through a development environment used by the internal application developers. Program managers as well as users, and developer experience experts, maintain contact with the user organisations and partners.

In Microsoft Dynamics the development is split in framework development and application development, where the framework development provides the runtime and development environment for the application development.

After becoming part of the Microsoft foundation, the technology stack is slowly exchanged. This, in turn, results in the possibility and sometimes also the need to negotiate features with the visual studio development organisation (see also Section 4.3.2).

SIM

The software products that SIM provides need to be substantially configured when used for consultancy projects or as part of water management systems. A substantial number of users who configure and sometimes customise the products are members of SIM as well. At the time of the interview the software product development was separated from the 'solutions' unit. Some of the interviewees expressed concern that this might cause SIM to lose its 'cutting edge,' as the innovations of the software products is perceived by some to take place through the consultancy projects.

UIB

Due to the open source development process, the users are part of the development process: with their participation in the testing of the releases, they contribute to the quality of the product. The users, system administrators in different organisations, also share implementation scripts handling background installations of

specific programs. These procedures are shared with the whole community.

The software itself depends on other open source projects. As it is supporting system administration, it is also dependent on developments in the underlying operating systems. Changes, especially in the operating systems, trigger changes in the software.

4.1.2. Keeping connected with the context of use

All of our interviewees emphasised the users and in some cases the implementers or customers who finalise the product for the users as the main sources of requirements and inspiration. However, users and customer contacts are not the same, as is the case in contract or project development. The release manager from Microsoft expressed his opinion: 'Developing for an unknown number of users was also mentioned as the main challenge of product related development compared to custom development.' So how do product development companies keep in contact with their customers and users? How do they understand the evolving needs of their users? That differs from company to company, depending on the overall business model.

UIQ

UIQ is owned by part of its customers and clients, high end mobile phone producers. As co-owners of the company, the clients have a rather powerful position when reviewing the contents of a new release. Prior research indicates that the tight connection to the customers even influences which methods are applicable in UIQ [52]. This seems to become problematic for the overall quality of the product. As phone producers compete with new features, they prioritise new features over improving the quality of the existing code base. Requirements, based on end-user tests and research, are sometimes difficult to communicate and prioritise when competing with new features.

To also bring the end-users perspective into the development process, a team of interaction designers work with a network of users. Through this network, a long term contact with pilot users or early adopters is maintained, which allows both the expert testing of individual features and ethnographically informed long term use studies. The interaction designers who seem to act as an advocate for the end-users have to convince the product management organisation so that they can convince their negotiation partners on the customer side.

Microsoft Dynamics

In Microsoft Dynamics, 'program managers' – who outside would be called product managers – are responsible, together with 'user experience' researchers, for the input by users, customers and customisers.

> '[...] it is the responsibility of the PMs [Program managers] basically. That is what they do. And the sources are of course direct interaction with customers, direct interaction with partners. But also [another role] has direct contact with individual countries, and individual countries [with] various sub-PM oriented [around the] product in that specific sub. And he is collecting requirements and pushing them up-channel and then they get prioritized.'

User experience and developer experience research – the latter focussing on the usability of the development environments – involve both observation of the usage of the software and communication with users and customisers for the interaction design of the specific interfaces. The practices of the customisers documented in [20] provide central criteria for the development environments, which rule out a number of possibilities for technical implementation of the necessary flexibility [56].

Developing for an unknown number and variety of customers and users was mentioned by our interviewee as the main challenge of product related development compared to custom development. The feedback cycle, getting a reaction from the customer, is much longer than in custom development. Also, the danger of being unaware of the consequences of taking out features that are important to some customers and users, is mentioned as a challenge.

SIM

The commonly emphasised importance of the usage influencing the development is expressed by one of our interviewees as follows:

> '[...B]asically, MIKE 11 which is our best selling software is [...] developed as part of some big projects we did in Bangladesh. [...T]hat is what really brought MIKE 11 from a small modelling system to something which is very sophisticated and was tested on the biggest rivers in the world. So [...] what I have seen, when our software has really taken a big leap, it has been in connection with projects – big projects.'

One of the current drivers of new development and re-engineering is the growing use of simulation software as part of water management. Here, the software is used to frequently to re-compute predictions of the behaviour of water systems based on real-time measurements. These predictions are then used to take action if values are threatening to get out of bound.

One interviewee described the development process from an end-user developer's point of view: Hydraulic engineers at SIM are quite familiar with the software. About 5% of the projects require the implementation of new features. The hydraulic engineer responsible for the project and the customisations implemented the changes directly in the development branch of the software, and, just in case it was necessary – the customer had installed a recently released version – the changes were applied to the release branch as well. Especially at the end of a release cycle, some changes could jeopardise the quality of the released software.

In the new organisation, a 'release board' is designed as the link between the usage and the development organisation. The work of the release board is backed up with an annual 'Product Development Priority Process.'

> 'Based on how the different offices have performed in terms of sales, they get a certain amount of points they can put forward towards certain feature. So first they come up with five features each, those are then put in a long list, and then it is sort of like the Eurovision Song Contest, you can put your points on all these different things. And the reason why we have done that, is because we have had a lot of complaints over the years. That they said: "We suggest this, we suggest this, and none of it makes it into the software." So we have put this into sort of a more structured framework, where we can follow it all the way through. And it is also good, because all the different offices, they start to interact a lot more, because they suddenly find common ground in terms of what can be usable for them.'

Even those in favour of the new organisation see the maintenance of the relation and feedback from the solutions department to the product department as one of its main challenges.

UIB

The inspiration for new features, beyond the necessary update to support new operating systems and hardware, is mainly grounded in the in-house experience with system administration. The in-house experience provides a frame for the understanding of the task and the 'rational' design of functionality. However, both customers and the open source community add to the requirements, as do discussions with potential customers at fairs. But

> '... we then have to understand: ok there are other companies where the requirements profile is a different one. That means for us that we have to flexibilise the whole time, that we have

to question our anticipation of how our software should be work in a meaningful way. Also this aspect has to be configurable.'

For instance, the following situation was reported: normally, a laptop should have all security updates installed before being active in a local area network. That does not work, if people are coming for a meeting and only have 5 min before starting up their presentation, which was the case for one customer/user.

As a main advantage for the contact with the customer, the short development cycles were mentioned. New features and changes can be implemented and released rapidly, providing the customer with a base to give feedback.

4.2. Architectures maintained by communities of practice

The architectural structure is regarded as an important asset in all companies: it is maintained, and upgrades are carefully deliberated and planned. Internally the architecture is maintained as a skill of a group of architects or lead developers. Important requirements for the architecture, however, originate in the distribution and coordination of the architectural design across different design constituencies. As the software needs to provide different possibilities to adapt the software, the interfaces for customisation and configuration pose important requirements on the architectural design. They both delimit the realm of, and bridge between, the different design constituencies.

4.2.1. Architecturing as skill of a group of architects/core developers

One of the more surprising issues was that architectural design of the products developed was not documented. This was the more surprising, as the documentation for the users in all cases had a high priority. The interviewee from Microsoft Dynamics gave a first hint how to interpret this circumstance. Comparing the product with which he was currently working with the development of a new product, he commented that for the new product every change impacting on the architecture had to be checked with the architects. In the Dynamics Nav development, the architecture is established, and the developers involved know it, so there is no need to develop and maintain an explicit documentation of the architecture. Architectural knowledge was taken care of by the assembly architects or senior developers, often developing and maintaining the structure of the software while interacting with the team. The understanding of the architectural design and how to use it and improve it is part of the tacit knowledge a developer acquires during the process of becoming part of a product development team. In the case specific discussion below, the notion design and architecture are used in line with the interviewee's usage of the terms. If not qualified, use notion of design refers to the technical respectively architectural design of the software product.

UIQ

Asked about the architecture, the UIQ interviewees could not point to a single document or a structured process. The architectural knowledge is shared by a group of system and applications architects.

Asked about who designs a new feature and where a feature is to be placed, the UIQ interviewees explain that dependencies and 'placing' of functionality is handled based on expert knowledge of the product managers and the architects. This is an issue both regarding the border between the framework and the application suite and regarding the interface between one's own and other's code.

> 'I think the people that are evolving ... a component, they know exactly [...] where the line is between Symbian work and UIQ platform work ... or where the line should be. I know that we have sometimes given code away to Symbian because its ... They hadn't time before to do it [...]. [W]e coded it ourselves, but now [...] we give this component to Symbian, because [...] every body knows it should be on that side of the line. So it is better for us as well, if we don't have to maintain that component. And the same is happening on the other side also, from the UIQ platform to our customers. Sometimes they give us components, if it is something that belongs to our platform it is better that we have it.'

The members of the group, experts in different cross-cutting concerns – like architecture, test, interaction design, and so on – serve as common resources for the different development teams. Architecture is one of these cross-cutting concerns.

The framework is, however, documented to support its users, that is, the developers use it for application development in the form of a style guide and a 'How to' document. Both contain information on the design rationale and the architecture.

Microsoft Dynamics

The Microsoft Dynamics interviewee first does not mention the architects as an independent role. Asked explicitly, he describes them as 'free floating' and negotiating with the developers over design. The character and age of the product was held responsible for this very informal practice. The architecture is not documented but shared as tacit knowledge among the group of architects.

This is supported by research on the customisation practices documented in [20], that indicates a steep learning curve for newcomers working with customisation of ERP systems.

SIM

At SIM the documentation of the software and the quality of the architectural design differ across the product suite. The development of the newer simulation engines have profited from the earlier developed ones. As the Graphical User Interface modules are developed by people with a computer science background, they tend to be designed more according to the book and also to be better documented.

The majority of the employees have a background in hydraulic engineering and have learned to program by customising the software. Newcomers learn by being part of development projects and by asking experienced members or the team. New features are designed by informal discussion at the whiteboard. The problem is, however, that this practice does not scale. It takes a new developer several years to understand, e.g., the code base of especially the simulation engines to be able to implement changes in a non-disruptive manner. See also [59,61].

Problems with this practice have motivated cooperation with the university, and in a current project the one-dimensional engines are being reengineered into a more generic form which can be used by the different applications. This re-engineering project is also a trial to implement more explicit architectural practices and to develop an architecture that allows isolating the most common types of customisation into modules that can be integrated in a plug and play fashion.

UIB

The knowledge around the design is shared by a few main developers of the software, most of whom sit in the same room. Meetings and discussions are the main measure to coordinate the development and the design changes. 'Well, it is not only my head but there are also other heads, and the systematisation takes place in the meetings.' The design of new features takes place 'on the whiteboard.' One problem for the interviewee here is that the results of discussions at meetings are not easily documented.

> 'Well, it is important for us, that the people look at each others' code. On the one hand to help each other, at the other hand to make sure that we are not all 'hanging in the air' when someone

falls sick. And somehow there is a chance of finding errors. ... and especially developers of neighbouring modules can take advantage of looking in the other modules code, to see whether it is actually implemented the way they imagined.'

The lead developer is responsible for release planning, scheduling, and budgeting. Two developers take care of the central modules, and working quite independently with the architecture, the lead developer is '...glad if [he] now and then get[s] to know about it in time'.

At the time of the interview the first thorough rework had just been released. The update addressed the architectural design and a redesign of data structures and persistency layer. The company also had help in the form of a Master thesis student, who did not have 'any emotional attachment to specific code parts' and could therefore address the redesign in an unbiased way. The Master Thesis provided a good but already out-dated documentation of the overall design. 'Of course the first design of a data structure does not keep long in practice.'

4.2.2. Interfaces separating and bridging the different design constituencies

Different design constituencies are not only separated through their different purposes and different practices, but often also through the technical design; all four products considered here provide configuration interfaces based on a layered design. However, the technical as well as the organisational borders between the different design constituencies are subject to continuing contest and maintenance.

UIQ

In UIQ's software, the borders between the application and the framework versus the framework and the operating system are implemented as Application Programming Interfaces. However, as the quote from Section 4.2.1 shows, local priorities and constraints sometimes lead to a blurring of these APIs; functionality belonging, from a design perspective, to one organisation is implemented in another organisation. Such code 'belonging somewhere else' is later handed over to the right organisation.

At the same time, the split requires cooperation and knowledge sharing across the organisational borders: As the two organisations originated in one, the coordination and knowledge sharing between UIQ and Symbian is well established and organisationally supported. The interviewees talk about having 'windows' into the other organisation, even on a role specific level.

'And the software developers also have windows in to Symbian when they have to discuss on detail technical problems as well. And we also put requirements on Symbian. Which is also what product management [is] doing. ... Because we want Symbian OS developing in the same direction as we see it, to have most gain for us of course. So we also have workshops where we try to convince them do as we... to go into the direction that we see will come and be important, like graphical effects and that stuff.'

The same is the case between UIQ and its customers. 'We have connection to customers on different levels. So the interaction designers also have windows to interaction designers on our customers' side. And on the business development level, we also have connections where we can discuss. We can set up meetings and point to people [...] to discuss new hardware roadmaps, and so on that, we have to know about.'

Microsoft Dynamics

In the Dynamics Nav software, the development environment implements the interface between the framework development and the application development. The development environment is also used for customisation outside Microsoft Dynamics. Development of application and its customisation takes place by defining objects through a modelling environment, complemented by code units in a proprietary programming language C/AL. The interface here is designed as a programming language. The configuration interface provides possibilities for implementation consultants at the partner companies and for experienced end-users to adapt the functionality through a runtime interface. As discussed in Section 4.1.2, program managers and user experience experts are involved in bringing requirements and inspiration from customers and partners to bear on the development process.

Within the development organisation, two rather distinct and stabile groups of developers are distinguished: framework developers and application developers.

'I think the normal is either you are an app developer or you are a framework developer. Simply because it is two different worlds. [...] The app developers are operating in C/AL– which is a higher level language which is more business oriented and easier to use – whereas a framework is typically either C++ or maybe C# but – like more complicated typically more technical language. And also if you are building app stuff, it is more important to understand what the customer needs. If you are building framework, it is probably more important to understand the possibilities in the technology. So from that the perspective it is two different sets of people.'

However, '[s]ome of the very senior people who has been [here] when it was a small company and not so many people, [...] they were doing whatever was needed to be done. Now it is just a big team, so now people specialize in areas.' These senior developers seem to be the ones who take on the role of the architects, making sure that the framework and the application develop in a synchronised way.

To coordinate the application development and the framework development, Microsoft Dynamics tracks dependencies between the two layers of the application. Such dependencies are also used to negotiate with the departments developing the tools and middleware the framework is built upon. 'So actually, we have been driving dependencies into Visual Studio for instance.'

SIM

To simulate a specific water system, the simulation software has to be configured by defining a model of the water system with the help of an application normally called 'the GUI'. If the existing modelling elements provided by the software are not sufficient, the software is customised by the consultant himself or a more technical inclined colleague changing the source code. The majority of the hydraulic engineers are able to program in the simulation engines. The changes traditionally became part of the current development branch. This resulted both in problematic code quality and in lack of funding for necessary substantial re-engineering.

The change of organisation at the time of the interview that separated the product development from the consultancy departments was partly motivated by the need to consolidate the software development and design of the software products. The re-engineering of the two different one-dimensional simulation engines, which in the previous organisation were evolved by two different business units into one common engine, was one of the first of these consolidation projects.

At the same time all interviewees were concerned to lose the contact between the innovative projects of the use context and the product development.

'The thing is we are also need to make sure that the project departments [...] have actually got the capability to do some of these changes. [...] When I am strictly speaking from production point of view, we are very interested in them building up this expertise as well. Because we would like to have some

feedback, we would like to get some good input in what makes it into the engine, what's required in the years to come. Because they have got this project view, that we don't have any more.'

Our interviewees emphasised the need of a better architecture in order to decouple project specific customisation from the main development branch. On the question, whether you could not isolate customisations he answered:

'Not yet, because we don't have that architecture right now. So the only way to do it, is to take that particular part of the code in this case part of the engine, take it out at a certain point in time, and do your customizations and then – unless you can agree with the software guys to merge them again – then you are going to be in trouble, because you have your own track. And that track is totally decoupled from the main track, but it depends on the main track, so it will not work.'

'Our idea is that we will make the engines open, so that the projects can do add-ons, customizations of the mainstream software without actually [...] messing with the main code. That's our ambition, but that's a big thing. I don't think that our management really recognizes how hard that is. It is something that will take a lot of time and cost a lot of money.'

UIB

UIB provides two interfaces for their users to adapt the software to the specific context of use: A configuration interface allows adjusting distribution policies, and for each combination of program and operating system an installation procedure has to be provided. If adaptation beyond the provided interfaces is needed, the software has to be changed. In Section 4.1.2 we have cited an example for such a feature request. This dynamic is accommodated through the agile process that allows for frequent releases.

The separation of adaptation and development is, however, responsible for the need to communicate across that border. The open source community is one of the important communication links. The in-house expertise in system administration allows the company to bridge between tool development and system administration practices.

4.3. Cycles within cycles

In all four cases, the interviewees discussed more than one rhythm defining the development process. The rhythms were related to different rationales either originating in the use context of the product or based on the necessity to revise the technical design. All four companies reported difficulties regarding the balance of different qualities in the development process. Three of the companies tried to address challenges by changing the structures of the development process.

4.3.1. Combining different rhythms to juggle different needs

For four of the four companies in the study, the interviewees identified different rhythms in the development process that was used to consciously structure the on-going work. The term 'rhythm' was used in two interviews without the author providing the term. One of the interviewees called the central release cycle the 'heartbeat' of the development process.

UIQ

UIQ releases two to three major versions each year. Between these releases, UIQ releases updates on previous versions containing bug fixes and the like. These minor releases might take place every two weeks. Each minor release includes requirements regarding new features, but also includes bug fixes and change requests. The requirements for new versions come from the customers owning the company and sponsoring the project. On top of that, the interaction design team gathers requirements and generates new ideas based on their work with their network of mobile phone users. Product management negotiates the requirements with the customers. As the mobile phone market is very volatile, new features and time to market are of high importance, which also implies that customers are pressing for requirement changes late in the process.

To help this situation, UIQ plans to increase the release rhythm. If customers only have to wait a few months until a feature is implemented, it might be easier to convince them to postpone the feature for the next release.

According to the quality manager, one of the challenges at the moment is to be clear about what can be implemented with what level of quality in a certain time frame.

Apart from new developments, UIQ maintains several old versions of the framework and applications suite. The development teams have to take care of those as well.

Microsoft Dynamics

The main rhythm of the development is defined by releases, which at the time of the interview were each taking between three and five years. Since then, the time per release has been shortened. A release requires an elaborate quality assurance process before distributing it to the public, e.g., testing for security holes. The interface is checked for offending language or bitmaps, documentation has to be in place, and so on. With a '.0 release' the whole product is shipped. More comprehensive changes like the implementation of a role tailored client might be distributed over several releases. Between two releases, service packs are released, which means that the whole product is updated. 'Then you have hot fixes:' that means fixes for a specific customer or a specific installation. Whether a customer specific fix is implemented or whether the fix is postponed until the next release depends on how important the fix is for all customers and how important the customer is who has the problem. 'Critical Design Change Requests' are 'hot fixes' of more importance, e.g., related to specific legal requirements. The latter normally have to be fixed within a certain timeframe. 'Hot fixes' do not require the same final testing before being distributed, as each customer can choose whether to implement is or not.

Internally, a release is structured according to a release process, which distributes the release in an initial phase, where requirements are collected and features for this release are specified. After that phase has ended with a Milestone 'M0,' a number of development milestones follow. For the ERP systems, the number of milestones per release varies between five and seven. In the current Dynamics Nav release there are seven milestones planned. The development milestones are followed by a stabilisation phase, after which the software is 'released to localisation,' where language and other country specific adaptations are made. During the specific release that was on-going when the interview took place the release management tried to organise the development so that the localisation could take place parallel to changes to the framework improving performance. In a similar way, the development milestones have been overlapped: different phases within each development milestones require different competences, thus developers can work with implementing the next milestone, while testers are still working with the previous one. In this way, one can press more than two milestones into a year.

Each milestone takes between 20 and 26 weeks. The first eight weeks are used to detail the design of the features assigned to the milestone ending with a 'Design Complete Tollgate.' Eight weeks of development follow, concluded with the 'Code Complete' tollgate. Testing starts parallel to the development, but is then the defining activity for the final 10 weeks of the milestone, with three more tollgates, the first is 'Full Test Coverage' the other two are defined by test metrics: how quick a bug is resolved after being found.

The two layers of the software – application and framework – follow the same rhythm, as normally features require changes to

both parts. However, within a specific release, application respective framework tasks might be scheduled to best support the objective of that release. 'If you were running an App[lication] centric release you obviously want to frontload the framework piece and make sure that is done and freeze that and then run the app depending on how…' The, in that case minor, changes to the framework are taken care of first to be then able to focus on the implementation of new application features.

The milestones are normally scoped so that they contain about eight weeks' development work. That way the availability of developers defines the scope of what can be developed within one milestone. The interviewee proposed to scope the milestones so they contain '60% commits' – features that are important – and two groups of 'decent targets' – nice to have features.

> 'And we would make a schedule that sort of tries to put commits in the first part of the milestone and of course targets later on. And if we then slide and get delayed, we simply start taking out the targets one by one […] Until of course we hit a critical … where you actually start hitting on commits then we have to re-evaluate: is this milestone time-driven or is it feature-driven.'

However, how features are split over the different milestones also depends on dependencies between changes to different parts of the system. Managing dependencies is therefore an important part of planning. 'Dependencies' are also the way requirements to other Microsoft software are formulated, and development is coordinated between Microsoft's middleware products, e.g., the database software, and the ERP system development.

SIM

At the time of the interview, SIM was in the midst of a change of the organisation and also the development process. The old development process was presented as follows:

A new version of the software was released every one to two years. Between these major releases three to four service packs were released. In the water resource software department about 15 people worked with development. The planning for the development followed a half-year rhythm. 'We would typically have two bigger projects, which maybe had a long-term focus, one or two of those. And then we would have a couple of two or three small projects which maybe had a duration of two or three months or so. So I think that we typically had maybe five development projects going on.' The bigger projects address also architectural improvements that allow decoupling the mainstream development from project specific customisations.

> 'So for instance, we are working a lot with isolating data access from engines and from interface. […] So we had some development rules, guidelines for how we develop things. So all our new software have a better architecture than our old legacy code.'

The projects themselves did not have a very formal structure.

> 'No, that was quite agile, and not very formal, that's true. […] We defined projects of course and then we established a project group, and that was it. So in that respect … Of course we had a formal way of doing that. But it was not very formal. There was not a lot of structure in our development process, which are probably both good and bad.'

The process around smaller, unplanned improvements that are initiated and paid by a project, but get included into the main software is described as follows:

> 'Yes, often the client, they describe their need. They describe kind of a use-case, […] this is the way we want to do this, and then we would write a small note […] on how we would implement that, like a small vision scope document, which we would of course discuss internally. We would try to make generalised things, so we could use them more broadly. We would always prefer to put them into the mainstream software, because if we could not do that it would be impossible to maintain, and nobody wants that. So we always try to make things as general as possible, so it could be used by others and not just that particular client, but it was often like the client has a need and then we discuss his needs internally and find out how we want to code and then we write a small proposal for him a vision scope document, and we say this will cost you 20,000 USD … whatever …'

In another interview, the pros and cons of the development process so far and the transition to the new organisation and processes were discussed:

> 'Well, I think there is a trade off, because … yes we have tried to be very agile previously, but the problem, I feel with being very agile is that sometimes you drop something on the floor because suddenly something new comes up, and you start focussing on that, and you don't get things finished off. So we […] still try to be fairly agile in terms of: if something serious comes up, we will focus on that.'

With the new organisation, the basic release frequency is not changed. However, the content of the release is controlled by a board, deciding also on ad hoc requirements from projects. Development and bug fixing is separated: The release-oriented development is interrupted for a period of two weeks to implement these minor issues and test and debug them. 'And so far, we have been through one service pack doing issue it like that. And all around we have felt that was a very nice way of doing it so'.

> 'So that is the more long term perspective; we would like to back that [release development] up of course with a sort of track, that runs with development, which are beyond the here and now release, and that is in the making. […] Right now it is really a production towards the next release.'

UIB

At the time of the interview the first thorough rework just had been released. The update addressed the architectural design and a redesign of data structures and persistency layer. The interviewee called this a major release. Similar major releases happened every two years. This update was financed through a customer contract.

Minor releases take place about every half year in connection with relevant software fairs. The fair schedule provides the 'heartbeat' for the development. Here a number of smaller bug fixes and developments are bundled and advertised. However, on the open source forum, changes and updates are published about every two weeks. These frequent changes are partly due to the character of the software. The software has to handle installations on different operating systems and different hardware platforms, and it is dependent on the operating system and system administration software on the server side. New operating systems and hardware require constant smaller and bigger updates.

Release planning is mainly done informally. Ideas for new features are collected internally, from customers and from the open source forum. The development lead sorts them, decides on what is affordable and schedules the new features for a specific fair. The overall schedule and progress are discussed in the weekly company meetings. If more specific discussions e.g., regarding design and dependencies become necessary, they are deferred to 'developer meetings' taking place about every three weeks. The break-down of new features takes place at the whiteboard. The majority of the planning and coordination takes place in an agile fashion.

4.3.2. Conflicting drivers of the evolution process

Most of the products experienced different drivers for their development process sometimes matching the different rhythms. From the interviews we can identify at least three different drivers for the development:

Users and customers request new features either motivated by developments in the use context or in order to compete in the market. Here, one can distinguish new features which normally are related to the main development cycle, and bug fixes and minor improvements that are bundled in service packs or minor releases. Changes in the base software, as well as in the software and hardware environment of the product, can be identified as a second class of drivers. The last category is improvement of the technical design and exchange of base software. This often relates to changes implemented over several releases talked about as a major revision.

The conflicting quality criteria – more functionality, technical innovation, consistency regarding architecture and user interface – are juggled by the process model discussed above where different rhythms are often combined. Other means are the organisation of the development teams and the flexibility of software 'outsourcing' part of the development to configuration and customisation.

UIQ

At UIQ, different drivers pull into different directions: Due to the volatility of the mobile phone market, time to market and new features have a high priority. As the interviewees admit, this sometimes leads to low code quality resulting in a lot of fixes. The recent change to a module centric organisation of the development – yet another trial to counter the negative effects through a suitable development structure – is described in the next section. However, our interviewees are concerned about a module based organisation; the danger is that cross-cutting concerns like architectural design, user interaction standards, and consistency are neglected. As another measure to address this situation, UIQ considers increasing the release rhythm. If customers only have to wait a few months until a feature is implemented, they might more easily convince postponing the feature for the next release, instead of pressing the development team and possibly jeopardising quality.

Microsoft Dynamics

In the interview with Microsoft Dynamics three different drivers for the evolution have been mentioned: The market, new features, bug fixes of different priorities, and the change of the technology stack.

> 'But also looking at ... the market and competitors [...] where are the trends going: So for instance 5 – 8 years ago we had the big CRM thing where everybody had to have CRM and all products were implementing CRM. [...] So now it is something new: It is integration with SOA or components or what have you like more loosely coupled. [...] So I think the requirements are driven from many places. We are trying to categorize based on customer input.'

The releases combine both new features and the innovation with regard to technological base. Here the interviewee refers to the release rhythm as a cause

> '... so like are you in a rhythm where [...] you have to ship a new set of features every three months. Then it does not really matter how big the release is. Of course it needs to be new stuff in there. But you can actually get away with it ... Whereas if you are releasing can be a shrink wrap product, it will take 3 to 4 years before the next one arrives, two years at last. Then you'd better make sure that the stuff you have in there actually is what's critically to the partners so they can actually get a very good revenue.'

Over the last releases, Microsoft Dynamics started to exchange a proprietary development technology stack bit by bit with a standard one. This had to be distributed over several releases.

SIM

SIM has been fighting with contradictory development drivers and quality criteria. The reorganisation of SIM is, by our interviewees, attributed to the balance between innovation and technical quality tipping too much to the innovation side:

> '[The developers] were really bombarded by [...] small requirements from projects. So it was difficult to keep focus, maintain focus on the long-term important stuff. Instead having all these small things that came in all the time. That could both be [...] feature requests from projects, it could be bugs, because our quality was not good enough. But they had a quite dynamic working day – you could say.'

From a technical design point of view, this practice has its advantages and disadvantages. In some cases, projects paid for improvements so that all future projects could take advantage of them.

> 'So when there is a customer who comes and says 'I would like to have vents in the software!' then you say: 'That's fine.' And you are happy of cause, because you have a customer that pays for some development that everybody can use. And so you are of cause interested to implement it as a general feature.'

The other side is that the main source code branch contains code that is specific for one past project and not used by anybody else.

> 'There are examples of features that are implemented in our software that are so specific that nobody else can use them. [...] Things that are implemented for projects 15 year ago, where big chunks of code are part of the software that can be activated if you do things in a specific way. [...] Those are examples of what you should definitely avoid.'

The new development organisation with a more rigour release planning discussed in the subsection above is geared towards protecting necessary technical consolidation.

UIB

UIB just released a fundamental redesign and consolidation of the server side implementation. The redesign has been an important step to develop a sustainable structure. 'This way we have a new conceptual design, and we hope that it will somehow hold at least the next three years. And so far it looks as if it would be extremely flexible.' This redesign was possible as a Master Thesis student in computer science could dedicate a substantial part of six months to the task. Additional funding came from one of the customers; the redesign also included a new persistency layer. However, the documentation of the new design in the Master thesis was already out-dated when the new version was released.

New features are developed for trade fairs that take place about every six months. The software slowly develops as an alternative to using fully commercial products in small and medium size enterprises. With the wider use, new requirements are popping up. Some of these features cannot be financed as part of the on-going business, and need financing by the customers. Here, UIB experiments with different models.

The final driver is a result of new releases of software to be distributed, hardware to be distributed, software and hardware on the server side. Here smaller releases are necessary to keep the software up-to-date. The development of installation scripts by the user community can be regarded as a way of 'outsourcing' part of that development to the usage/customisation context.

During discussions the need to juggle the different quality dimensions became visible: The company shares the value of the open source community for sound technical design. In case of delays, announced features for the next release might be skipped, rather than taking in 'ugly code' solutions that have to be replaced later. However, if an important customer insists on a high priority feature, short cuts might be taken.

4.3.3. Developing organisational structures to balance different qualities

Based on the development cycle and the business model, different quality requirements have to be balanced. All companies experienced the difficulty to balance use quality, quality related to the development process, and product related qualities. The quality requirements were balanced in different ways. Two of the organisations reported a major organisational change addressing a perceived imbalance. The largest company consciously changed the organisation of the team based on the character of the current release.

UIQ

The internal organisation had recently changed from a project based organisation where project groups were recruited from competence centres according to the needs of each of the customer funded projects, and moved to a more permanent division into groups around specific parts of the application or framework. The motivation is that each group develops specific expertise for their part of the application. Other motivations for the reorganisation were related to quality assurance. In each group of 15–20 people, product managers, interaction designers, software architects, developers, and testers work together and report to the same project manager.

As a drawback of the new organisation, the lack of maintenance of skills is mentioned across modules, as well as the interaction designers missing their community of practice. Knowledge sharing across module centred development groups is one of the purposes of a systems design group retaining senior employees from different software engineering sub disciplines: software architecture, interaction design, product management, testing.

Another cross sharing group 'looking into the glue between different applications' is the system design group. One example that is a cross sharing concern is performance. From an interaction design perspective, another example is 'a similar look and feel between the applications even if they are designed by different teams. [...] And also [...] to have a similar architecture, or to have somehow an architecture that is.'

The members of the group, experts in different cross cutting concerns – like architecture, test, interaction design, and so on – serve as common resources for the different development teams. The person who works with ethnographic user studies takes care that the network of pilot users is situated at the system design group.

Microsoft Dynamics

In Microsoft Dynamics, the organisation of the development team depends on what is the main goal for the current release, for example, the previous release was organised in groups based on modules, as part of the technology stack was exchanged. For this release, the development team is split into feature teams. Each feature team has 1 or 2 program managers, a number of developers and testers, competencies from user experience and documentation experts.

The two layers of the software – application and framework – follow the same rhythm, as features normally require changes to both parts. However, within a release, the work on the framework and on the application might be scheduled depending on the character of the changes required. 'If you were running an App centric release you obviously want to frontload the framework piece and make sure that is done and freeze that and then run the app depending on how...'

As a new feature for a well-established product touches on different parts of the existing software, developers who know specific parts of the system well are needed in different feature teams.

> 'If you are working on a (...) 1.0 product then it is easier to have very distinct teams because you have got a brand new thing, so you can define your way out of it. In Nav it is like a legacy product that has been here for many years with tons of functionality and stuff has been developed and people have left and come in and ownership has shifted and so on. So it is difficult to maintain a strict and lasting ownership. For instance you have people who have been here for like 10 years. And they know all about the product in this specific area. So they will typically have a lead role for several teams.'

Day to day knowledge sharing is also supported with practices borrowed from agile development: 'But in the end we are also like adopting pieces of the Scrum methodology because they have some attributes that we can use. So like burn downs and burn ups and the whole concept of having sort of morning meetings and stuff like that. Those have been incorporated.'

SIM

The new organisation at SIM is introduced to balance pressure for new features with better quality of the development. A number of committees have been introduced to make the decision process over development tasks more accountable and provide a better base for prioritisation. A service board prioritises on issues like bugs or project related features to be included into the next service pack. A release board decides on what to include in the next release. Thanks to changes in the accounting pattern, the fixed cycle of one release per year can be prolonged if serious re-development is planned. At the time of the interview the technology roadmap group was about to be established, whose responsibility it would be to outline strategic and innovative development spanning more than one release.

UIB

UIB's development process is mainly oriented towards the functionality and feature requirements from the user and customer community. The open source business model, on the one hand, does not provide direct revenue for the company; on the other hand, the community contributes to the development by sharing installation scripts and by contributing to the quality insurance of the product: The developers, of course, test their development. Thereafter, they ask in-house system administrators to check a new feature. When a feature is published to the open source community, the members in the community download and use the new version and report the errors. The company is eager to correct bugs. The interviewee is aware that this practice is neither ISO 9000 nor CMM compatible. However, the open source community process is sometimes accepted as an accountable quality assurance process.

Another issue is the financing of major revisions or new developments: As UIB does not sell its product but distributes it as open source, there is not much of a margin for such projects. For bigger development tasks, which are too comprehensive for pre-financing or for a single customer, UIB has started to work with co-financing by the community. Customers can 'buy in' and contribute with others to the development that – when it is fully paid – is published to the open source community. At the time of writing, the first of this co-financed features is implemented and two new ones are set up.

5. Software development beyond the project

The analysis above shows that sustaining a software product is not an easy achievement. Several of the reported practices are not

aligned with promoted software engineering methods, tools and techniques do not support the development and evolution of software products. This discussion section highlights the specificities of the practices reported by our interviewees and the challenges they provide for the software engineering discipline.

As mentioned in the introduction, the discussion also provides references to relevant literature from different fields. Each of the subsections below raises the issue, refers to the cases for empirical evidence, discuses relevant literature and summarises the main points. Table 2 summarises the analysis for a quick reference when reading the discussion.

5.1. Intertwining practices of design and use

All of the ecosystems in Section 2 indicate that software products of the more comprehensive kind discussed here are developed through interlacing heterogeneous development and use practices. To talk about *the* development versus *the* usage does not match the reported situation. This also implies that software product development needs to consider not only the needs of the user, but also the needs of companies implementing and customising the product to fit with the specific use context. In three of the four cases the dependency and interaction with developers of the products used for implementation were addressed explicitly. UIB partly lived from providing continuous adaptation of the software to new operating systems and hardware as a service. The user interface framework for mobile applications depended on the Symbian operating system. The technical interface and communication links were carefully established and maintained by UIQ, Symbian corporation, and the customers respectively owners. Microsoft Dynamics both keep close contact with the consultancies implementing the ERP system, but also aim at influencing the development of the tools and middleware needed for the development.

Prior research with software product companies provides additional evidence: Hansson et al. [28,29] report about a small company providing a locality booking system to municipalities and bigger sports places. They soundly claim – and the empirical research supports the claim – that the development is driven by the users and their needs. The communication takes place through the support line and through user meetings arranged by the company.

In the ecosystems analysed here, development takes place on different levels in parallel. There is neither one 'food chain' of innovation originating from the most basic technology and rippling down to the usage nor is the innovation constraint to the product developer and 3rd party developers. In 3 of the 4 cases, the developed product can be regarded as part of several ecosystems. E.g. the UIB tool extends the unix kernel as well as the targeted operation systems. The notion of developer and users points to relative roles in relation to one specific thread in the interlacing.

This challenges the research on software evolution as well as the current state of the art on use oriented design and development. In the research on evolution, the usage features as an abstract source of feedback from which new requirements appear. Only one of the chapters in [45] on software evolution and feedback presents research addressing: (1) the co-evolution of software and the usage within a socio-technical ecosystem, and (2) the need of mutual understanding of domain experts and software engineers as an enabling factor [47].

Lehman's insights in the evolutionary dynamics of software usage underpin the early texts on software process models. Boehm's risk driven development [8] and the Unified Process [34] open up for iterations that enable development teams to provoke feedback early in the process in order to deliver a system which represents a better starting point for the evolution thereafter. Use oriented design and development [24,66] organises the development around mutual learning cycles. The literature on these approaches, however, focuses on custom development, not product evolution. From a process perspective, agile development processes, such as XP [5], Scrum [55] or the Crystal family [14], implement a development process that only ends when the value of changes and new features no longer justifies the development effort. In our previous research we have seen product development companies whose processes resemble agile practices [28,29]. However, neither the research nor the textbooks explicitly address the challenges that the necessary interaction across heterogeneous design constituencies poses on the development process.

Research on User Driven innovation similarly falls short. The book on democratising innovation by v. Hippel [31] e.g. juxtaposes user and corporate innovation. In product development design and innovation takes place on all levels of the ecosystem. It is not a community of users that is innovating, but rather users, implementers, customisers, the development organisation and also the providers of the tools and middleware.

The analysis makes visible that the development at the software product providers needs to take the distribution of design, development and innovation across the ecosystem into account and that the resulting practices are required to extend the perspective of the software engineering methods, tool and technique development beyond the traditional frame of the project. In the remainder of this section we highlight the specific points where the software engineering has to be developed beyond the project.

5.2. Deferring design to 3rd party developers, organisational implementation and use

All of the products subject to the interviews provide configuration and customisation interfaces to allow for adaption of the software during implementation and usage. Different customisation and configuration interfaces and technologies are used in the same software product depending on the target implementers and users. Often one product offers different layers for further design by users and implementers. The possibilities range from changing settings in a configuration interface to the need to provide plug-ins and changing the source code of the product through a custom development environment.

Variability management, domain specific programming language and End User Development address the solution to the design problem from different angles. The provision of tools for generic programming [16], product line architectures [9,13], and model-driven development [6] all provide approaches to rationalise the development of software that resembles each other as a family of programs.

However, the intended user of these technologies is by and large a fellow software engineer resolving the variability prior to delivery and deployment rather than implementers and domain expert finalising and evolving the design. This might be due to that many of these approaches are addressing software development for technical embedded systems. Technologies supporting product configuration and customisation need to take into account that evolution takes place on different levels and by different design constituencies, which also means that customisations and configuration need to be carried over to new versions of the base product. Although programming language technologies and software architecture approaches in theory provide the necessary flexibility, they are not often used. This might have good reasons: Sestoft and Vaucouleur, for example, indicate that the existing technologies for making flexible the technical design of software do not support the specificities of the customisation and configuration practices around ERP systems [56].[3] Though the article refers to development

[3] For documentation of customisation and configuration practices see [20].

Table 2
Overview over analysis results.

Company	Product and characteristics	Interlacing design constituencies	Connecting to usage	Architecture knowledge	Interfaces between design constituencies	Development rhythms	Evolution drivers	Organisation of the development teams
UIQ	User interface framework for high end mobile devices Customers are owners	• Symbian operating system development • UIQ developers • Customers • 3rd party development • Users	• Owners and customers design new features and apps • Pilot user network • (3rd party developer community)	Distributed among a group of lead architects	• APIs between operating system and framework and between framework and applications	• Updates: very 2 weeks • Releases: 2–3 per year	• Bug fixes • New features	Module based, additional structure for cross cutting concerns (usability, architecture)
Microsoft Dynamics	Dynamics Nav ERP system Developed as a half product	• Framework developers • Application developers • Implementation consultancies • Customers and users	• User conferences • Implementation partner network • Pilot partners and pilot customer organisations	A group of architects participating in design meetings and discussions	• Programming language and modelling environment • Master data and configuration environment • Interface to middleware	• Bug fixes • Hot fixes • Releases: 3–5 years • Within releases 5–7 'milestones' • Changes in the technology stack can take more than one release	• Bug fixes • New features • Architectural revision (exchange of technology stack)	Depending on the focus of the release: can be module based or feature based
SIM	Hydraulic simulation software Scientific software (scientists as developers)	• Developers • In house users • External users • Water management system developers	• In house use • 'Bidding for features' • User conferences	Lead developers' tacit knowledge	• Modelling of water systems • Source code	• Bug fixes 3 times/year • Releases every 1–2 years • Re-engineering of architecture takes more than 1 release cycle	• Bug fixes • New features • Change of architecture	Product related small teams Recently organisational separated from in house use departments
UIB	System administration tool: Open source development Small scale development	• Developers at UIB • Open source user community	• In house use • Open source community • Fairs • Projects for customers	Distributed between 2 and 3 main developers	• Plug-in interface for installation scripts	• Continuous bug fixes • Minor releases 2 a year • Major releases: every 2 years	• Bug fixes • New features • Re-engineering to improve architecture	Small development team with in-house users also developing minor tasks

practices in a software specific ecosystem, the result indicates that the viability of a technical design depends not only on the, from a technical perspective, preferred qualities but also on how the design supports several heterogeneous actors in the software ecosystem contributing to the design through different development and tailoring interfaces.

Research in the field of End User development addresses the not only the understanding of the usage dynamics and support for tailoring by designing suitable configuration and customisation interfaces, end-user development languages and environments but also the environments supporting the cooperation around and sharing of appropriations. Refs. [43] and [41] offer two excellent overviews of the field. The development of the tailorable software is understood as meta-design, designing a frame for the design [23] – supporting both the diversity of usage and the evolution of usage over time – the intertwining of tailoring or end user development and professional software engineering is hardly addressed. (See [3,22,1] for exceptions.)

The architectural design for software ecosystems implementing interfaces for heterogeneous design constituencies fall between technically inclined research on programming language technologies and domain specific languages and research in the area of End User Development. Programming language technology, regards software developers as their audience and caters to development from scratch. The research on End User Development provides design examples for a number of interfaces and design environments catering to developers that do not have software development as their main expertise; thus far, there is no satisfying categorisation of these techniques regarding benefits and constraints. Half of the software products subject to this article support both professional developers and end user developers combining different techniques in the same architectural design. Most of the interfaces observed in real world ecosystems – be it APIs, development environments and programming languages or graphical interfaces – do not implement the principles recommended by research. Furthermore, the interfaces between the different design constituencies are contested and need to be maintained.

With other words, the empirical research presented here indicates the need for further research to address the complexity of providing designs and tools for heterogeneous design constituencies and supporting cross constituency collaboration.

5.3. Communication between developers, implementers and users

Communication and collaboration between heterogeneous actors across different design constituencies in a software ecosystem has been a central issue to all our interviewees. Software engineering normally aims at controlling such interaction in form of requirements and change management. Though agile development emphasises customer collaboration and the embrace of change, the interface to the use context is localised in roles taken on by individuals like the on site customer in extreme programming or the product owner in scrum. Though some of the interviewees voiced difficulties to deliberate changes and reserve development time for technical upgrade of the software, the main challenge our interviewees emphasised was not to protect the development from the influence of the usage but to ensure sufficient communication to keep the innovative edge of the software product. Even as the product provides interfaces for configuration and customisation, the base product itself needs to evolve to answer to changes in the technical as well as the business context.

Most of the interviewees reported both formalised communication fora and conscious use of informal communication channels. The former can be attained by: program or product management keeping contact (UIQ, Microsoft), a interaction design group hosting a network of early adapters (UIQ), UIB's open source community, or even a formal bidding process for new features and functionality (SIM). Although not mentioned in the interviews, Microsoft and SIM host user conferences where new features are presented and feedback and innovations from the user community are collected.

In the informal fora, the collection of feedback and ideas for new features does not take place systematically. The examples for informal but consciously deployed communication possibilities are: discussions at fairs (UIB), contact and communication with in-house users (UIB and SIM) or 'communication windows' into the customer as well as supplier organisation (UIQ). From previous research, the 'manning of support lines by developers' [28,29] is reported.

How then are the innovations in the use context communicated to the software product development process? Recently, the processes around generification of software, the processes through which the product development companies define the standard a software product should support, have been addressed by researchers from the area of science and technology studies and Computer Supported Cooperative Work (CSCW) [49,37]. Andersen and Mørch [1] report about the collaboration between product developers and end user developers with respect to understanding evolving requirements. Hess et al. [30] report from a software product company that systematically included representatives of their user community in the development. The systematically analyse the challenges of such an endeavour, due to heterogeneous interests between the stakeholders and resources in terms of time and effort that users are willing and able to invest. Such research is rare. Communication and collaboration between the product development organisation and the implementers, users and customers is normally addressed as requirements prioritisation and release planning [7] which is discussed in the next section.

The existence of both formal and informal contact and communication channels, as well as the emphasis our interviewees placed on this theme, indicate the importance of these channels for the continuous evolution of the software. The communication channels are means for aligning the parallel heterogeneous design and development activities.

Today, these channels are independent of the software product itself. A challenge for the End-User Development and appropriation support would be the integration of communication of innovation and design ideas with the software itself so that developers and users can refer to the software at hand in a direct manipulation manner when discussing, e.g., in a product integrated newsgroup about potential new features.

5.4. Managing an overlay of different evolution cycles

Although a main release cycle providing the 'heartbeat' of the development is easily identifiable in all cases, secondary development cycles could also be identified and named in every case. The drivers identified roughly fit the known categorisation of maintenance tasks into perfective and adaptive, corrective, and preventive maintenance [44]. However, that these different maintenance categories respectively evolution drivers motivate different kinds of minor and major releases following different rhythms came to us as a surprise. The coordination of these different cycles and the integration of the results have been subject to experimentation and discussion in the different companies.

One motivation for the reorganisation of SIM's software development has been to control the continuous stream of new requirements from the project organisation in order to be able to set aside time for consolidating the software products. Bug fixes and minor patches were time boxed so that they did not spread out over development requiring more concentration. The separation of

development and maintenance done in two different teams had led to quality problems, which was one of the reasons why UIQ reorganised the development into a substructure based on the module structure of the framework. This, in turn, created problems regarding cross module and application consistency and quality. The organisation of the Dynamics Nav development team depended on the characteristics of the current release. At UIB the dependency on paying customers resulted in a prioritisation of the features providing value for them. Without the (free) work of a Master Thesis student, the rework of the architecture and persistency layer would have probably had to wait. Hanson et al. [28,29] describe similar conflicts between implementing short-term requirements and necessary rework to upgrade the technological base. Such conflicts and experimentation indicate that the concepts and tools to manage software development do not provide the necessary support for companies to control and coordinate the different development cycles.

Research on product management emphasises economic and managerial aspects and does not address the software development processes. (See van de Weerd [63] both as an example and for an extensive literature review.)

If software engineering research addresses the intersection between software product development and software product management, it does so under the heading of requirements management. See [25,39] and especially [40] as examples. In a comprehensive qualitative study on requirements engineering in market driven software development, Karlsson et al. [38] identify the management of a 'constant flow of requirements' and the 'volatility of requirements' as important challenges when planning releases. Bjarnasson et al. [7] identify the operation in a rapidly changing market and the lack of awareness of overall goals as root causes for overscoping. The answer often is more rational and structured release planning as Zorn-Pauli et al. [65] propose. As an exception, Rautiainen [50] addresses agile release planning in a startup software product company. He proposes a scrum inspired process for the software product development process, combining strategic, release, development versions and daily built cycles and successfully introduces it. The author's 'temporal pacing framework' is designed to break down the product management roadmap into suitable release-, version-, and daily development cycles. Maybe due to the start up situation, the article does not relate these cycles to different drivers of evolution or to the technical re-engineering that seems to become necessary in regular intervals.

The majority of the contributions addresses the feature development cycle, the 'heartbeat' of the software ecosystems. As from a design and development perspective the 'debugging and small changes cycle' is regarded as a necessary evil and only addressed in [50]. The slower, technical re-engineering cycle is not subject of requirements handling or product management therefore might not show in that research. It, however, is crucial for the software products as it is necessary to sustain their evolvability.

Our interviews did not address development environment and versioning support for integrating the results of these overlapping development processes. Here, we see the potential for combining further empirical investigation and the development of tools and algorithms for versioning and integration.

5.5. Informal knowledge management practices

All of the interviewees report the software development relying on informal knowledge management practices; as the development team has worked with the application for years, the team members know 'their way around' in the source code. In the bigger development teams (Microsoft Dynamics and UIQ), user interface designers, product or program managers and software architects are examples of roles that take responsibility for the overall integrity of the user interface, the functionality and the technical design.

Similar informal practices have been described by de Souza and Remiles [57]. Hansson et al. [27] also report the important role of the lead developer in the evolution of the software product. With respect to the role of the architect, these findings confirm the results of [61] who based on a interview study with lead developers and architects of software product companies provide an indication that the tools and methods proposed by software architecture knowledge management that aim on a codification of architectural knowledge might not fit the needs of software product evolution.

The lack of documentation and the informal architecturing practices, however, are not unproblematic; Unphon [59] e.g., reports that the introduction of new developers requires a substantial amount of time and non-expert code changes can result in low code quality. Unphon proposes light-weight tool support for architecture compliance checking and light-weight architectural evaluation methods to support the integration of software architecture and architecturing in the everyday development practice.

Parallel to Unphon's argumentation regarding the introduction of architecture documentation, the design of suitable documentation practices needs to take into account that developers are familiar with the software and that the software and its requirements are changing frequently, and therefore the documentation must change accordingly. The development, maintenance, and usage of the documentation needs to be carefully embedded in the day to day work practice in order not to impact the work of the software architect negatively [61]. The recently acclaimed research on traceability and Model Driven Engineering [4] might provide approaches and technologies which can be deployed carefully based on in-depth studies of product development practices.

5.6. What then is the role of projects in software ecosystem evolution?

Projects do still exist. Projects provide temporary closure that allow actors to assemble a more permanent constituency and development around specific and often more comprehensive developments tasks, e.g., the UIB developed a subscription system to share the expenses of a major upgrade. For innovative and new developments, pilot users are recruited by Microsoft Dynamics [48]. Prototypes of major changes might be developed in a project organisation. UIQ chose to develop end-user functionality required by a specific assembly of costumers and owners as projects and in parallel evolve the framework. At SIM the merging and re-engineering of the one-dimensional engines is organised to resemble a more conventional project [59].

On the one hand, these projects are constructed by delimiting design constituency, defining the scope and assigning development resources. On the other hand, they are based on the innovation and design processes across the ecosystem, and their results feed into these processes again. Seen from the ecosystem perspective, projects are a specific form to organise development that is used under certain conditions for specific purposes and provides certain constraints.

6. Conclusions and future research

The main contribution of this article is the presentation of common features of product development and evolution in four companies. Although size, kind of software and business models differ, the commonalities are striking:

– The software products are developed in interaction with a product related ecosystem where part of the design is deferred to other actors closer to the concrete use context. However, real world ecosystems do not only consist of the product developer

providing a platform for 3rd party developers together with the product but includes several layers of actors customising and configuring the software product. Often the product can be regarded as part of one or several ecosystems itself.
- Innovation takes place across the whole ecosystem. Contact with users and other actors is therefore important to keep the innovative edge of the software product.
- The technical design and architecture exists as practice of architects and core developers rather than as explicit documentation.
- The interfaces for configuration and customisation both separate and bridge the different design constituencies. They are contested and need to be maintained continuously.
- To juggle different and sometimes conflicting development drivers – bug fixes, new features, technical re-engineering – the development processes consist of an overlay of different development cycles.
- As the companies have to balance different qualities of their products, they struggle with finding the right organisation of the development team. The organisation might change depending on different emphasis in the development, which, in turn, depends on the dynamics of the ecosystem.

The empirical material provides a rich bouquet of different practices highlighting their specific advantages and disadvantages. Much of what is reported is not supported by the software engineering textbooks, but makes perfect sense, considering that the frame of reference for product development is not a project – defined by a fixed scope, time and design constituency – but continuous innovation across the respective ecosystem. As software is increasingly developed, adopted and deployed in the form of customisable and configurable products, software engineering as a discipline needs to take on the challenge to support this way of organising software development with methods, tools and techniques. Our result indicates that software ecosystems challenge some of the very core assumptions of traditional software engineering.

The discussion highlights some of these challenges: keeping contact with other actors in the ecosystem, techniques to support multilevel development and evolution, managing an overlay of development cycles with different rhythms, and documentation and modelling support for continuous development. Projects, however, still exist as a way to provide temporary closure and are defined where they are deemed necessary.

As to a research contribution, this article is not designed to evaluate solutions, as this is impossible based on the research presented. Rather than jumping to possibly premature solutions, the article aims at providing an understanding of the practices addressed.

The research presented here is meant to indicate the need for further research in order to address the challenges presented. It opens up possibilities in different directions for future research. First, the analysis results provide inspiration for research focussing on specific method development, such as supporting planning for the overlay of several development cycles with different rhythms. Second, it can inform future research as to what to take into account when evaluating specific method development in practice, for example, how the proposed requirements triage methodology [40] influences the relation to implementers when applied to socially embedded software products. A third possible line of future investigation could be the design of surveys addressing specific aspects of software product development, thus providing more quantifiable evidence on the themes addressed in the interviews, such as the relationship between size and the lengths of the different evolution cycles.

Last but not least, the results presented here open up further investigation of the interaction across the whole ecosystem and the possibility of better supporting user-driven innovation of software products. The research presented here analyses the software development in software ecosystems from a product provider point of view. Other actors in the ecosystem have different interests and perspectives. To support collaboration around design and evolution of software products, all actors in an ecosystem need to be taken into consideration.

Acknowledgements

Thanks to our interviewees and the companies for being open and forthcoming regarding their development practices. Thanks also to Vibeke Ervø and Peter Sestoft for support and discussion.

Appendix A

Interview guideline example

- Personal Background and experience, role in the organisation
- History of the software
  - (projects as drivers)
- History of the organisation
  - Changes throughout time
  - Motivation for the last organisational change
- Development practices
  - Documents
  - Distribution of work
  - What kind of processes
  - What triggers a release
  - Service patches?
  - Usage of the product influencing the development
- Roles in the development (Formal/informal, User representation)
  - Responsibilities
  - Changes in future
  - Chief architect/Head of innovation
- Documents in the process
- User feedback
  - User community
  - Systematic collection

References

[1] R. Andersen, A.I. Mørch, Mutual development: a case study in customer-initiated software product development, in: V. Pipek, M.B. Rosson, B. De Ruyter, V. Wulf (Eds.), End-User Development. Proceedings of the 2nd International Symposium, IS-EUD 2009, Siegen, Germany, March 2–4, 2009, Springer, Berlin, Heidelberg, 2009, pp. 31–49. http://dx.doi.org/10.1007/978-3-642-00427-8_3.

[2] C. Andersson, Improving the KM in ERP Development, M.Sc. Thesis, IT University of Copenhagen, Denmark, 2010.

[3] C. Ardito, P. Buono, M.F. Costabile, R. Lanzilotti, A. Piccinno, End users as co-designers of their own tools and products, J. Vis. Lang. Comput. 23 (2) (2012) 78–90.

[4] M. Barbero, F. Jouault, J. Bézivin, Model driven management of complex systems: implementing the macroscope's vision, in: Engineering of Computer Based Systems, IEEE, 2008, pp. 277–286. http://dx.doi.org/10.1109/ECBS.2008.42.

[5] K. Beck, C. Andres, Extreme Programming Explained: Embrace Change, second ed., Addison-Wesley Professional, 2004.

[6] J. Bézivin, Model driven engineering: an emerging technical space, in: R. Lämmel, J. Saraiva, J. Visser (Eds.), Generative and Transformational Techniques in Software Engineering, GTTSE, 2006, pp. 36–64. http://dx.doi.org/10.1007/11877028_2.

[7] E. Bjarnason, K. Wnuk, B. Regnell, Are you biting off more than you can chew? A case study on causes and effects of overscoping in large-scale software engineering, Inf. Softw. Technol. 54 (10) (2012) 1107–1124. http://dx.doi.org/10.1016/j.infsof.2012.04.006.

[8] B. Boehm, A spiral model of software development and enhancement, IEEE Comput. 21 (5) (1988) 61–72, http://dx.doi.org/10.1109/2.59.

[9] J. Bosch, Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[10] J. Bosch, From software product lines to software ecosystems, in: Proceedings of the 13th International Software Product Line Conference (SPLC '09), Carnegie Mellon University, Pittsburgh, PA, USA, 2009, pp. 111–119.
[11] J. Bosch (Ed.), Special issue on software ecosystems, J. Syst. Softw. 85(7) (2012).
[12] CMMI for Software Engineering (CMMI-SW, V1.1), Staged Representation, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, 2002.
[13] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, 2001.
[14] A. Cockburn, Agile Software Development, Addison-Wesley Longman Publishing Co., Inc., 2002.
[15] P.E. Cordero Carballo, Management of Evolution in Large Software Systems, M.Sc. Thesis, IT-University of Copenhagen, Denmark, 2006 (confidential).
[16] K. Czarnecki, U.W. Eisenecker, Generative Programming: Methods, Tools, and Applications, ACM Press/Addison-Wesley Publishing Co., 2002.
[17] Y. Dittrich, R. Giuffrida, Exploring the role of instant messaging in a global software development project, in: 2011 6th IEEE International Conference on Global Software Engineering (ICGSE), IEEE, 2011, pp. 103–112.
[18] Y. Dittrich, K. Rönkkö, J. Eriksson, C. Hansson, O. Lindeberg, Co-operative method development – combining qualitative empirical research with method, technique and process improvement, J. Empir. Softw. Eng. 13 (2008) 231–260.
[19] Y. Dittrich, S. Vaucouleur, Customization and Upgrade of ERP Systems. An Empirical Perspective, Technical Report, IT University of Copenhagen, 2008, TR-2008-105.
[20] Y. Dittrich, S. Vaucouleur, S. Giff, ERP customization as software engineering: knowledge sharing and cooperation, IEEE Softw. 26 (6) (2009) 41–47.
[21] U. Eklund, J. Bosch, Architecture for large-scale innovation experiment systems, Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), IEEE, 2012. pp. 244–248.
[22] J. Eriksson, Y. Dittrich, Combining tailoring and evolutionary software development for rapidly changing business systems, J. Org. End-User Comput. 19 (2007) 47–64.
[23] G. Fischer, E. Giaccardi, Meta-design: a framework for the future of end-user development, in: H. Liebermann, F. Paternò, V. Wulf (Eds.), End User Development, Springer, Netherlands, 2006, pp. 427–457. http://dx.doi.org/10.1007/1-4020-5386-X_19.
[24] C. Floyd, F.-M. Reisin, G. Schmidt, STEPS to software development with users, in: G. Ghezzi, J.A. McDermid (Eds.), 2nd European Software Engineering Conference University of Warwick, Coventry, UK September 11–15, 1989, Springer, Berlin, Heidelberg, 1989. http://dx.doi.org/10.1007/3-540-51635-2_32.
[25] T. Gorschek, C. Wohlin, Requirements abstraction model, Requirements Eng. 11 (2006) 79–101, http://dx.doi.org/10.1007/s00766-005-0020-7.
[26] G.K. Hanssen, A longitudinal case study of an emerging software ecosystem: implications for practice and theory, J. Syst. Softw. 85 (7) (2012) 1455–1466.
[27] C. Hansson, Y. Dittrich, B. Gustafsson, S. Zarnak, How agile are industrial software development practices?, J Syst. Softw. 79 (2006) 1295–1311.
[28] C. Hansson, Y. Dittrich, D. Randall, How to include users in the development of off-the-shelf software: a case for complementing participatory design with agile development, in: Proceedings of the Hawaiin International Conference on System Sciences (HICSS), January 4–7th, 2006. http://dx.doi.org/10.1109/HICSS.2006.205.
[29] C. Hansson, Y. Dittrich, D. Randall, Agile processes enhancing user participation for small providers of off-the-shelf software, in: J. Eckstein, H. Baumeister (Eds.), Extreme Programming and Agile Processes in Software Engineering. Proceedings of the 5th International XP Conference, Garmisch-Partenkirchen, Germany, 2004. http://dx.doi.org/10.1007/978-3-540-24853-8_20.
[30] J. Hess, D. Randall, V. Pipek, V. Wulf, Involving users in the wild—participatory product development in and with online communities, Int. J. Hum.–Comput. Stud. 71 (2013) 570–589.
[31] E.v. Hippel, Democratizing Innovation, The MIT Press, London, 2005.
[32] IEEE Computer Society, Guide to the Software Engineering Body of Knowledge (SWEBOK), Version 3, 2004. <http://www.computer.org/portal/web/swebok/home> (accessed 03.11.13).
[33] IEEE Computer Society and ISO/IEC JTC 1/SC7, SEVOCAB: Software Engineering Vocabulary. <http://pascal.computer.org/sev_display/index.action> (accessed 03.11.13).
[34] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley Longman Publishing Co., Inc., 1999.
[35] S. Jansen, S. Brinkkemper, A. Finkelstein, Business network management as a survival strategy: a tale of two software ecosystems, in: Proceedings of the First Workshop on Software Ecosystems, 2009, pp. 34–48.
[36] S. Jansen, S. Brinkkemper, J. Souer, L. Luinenburg, Shades of gray: opening up a software producing organization with the open software enterprise model, J. Syst. Softw. 85 (7) (2012) 1495–1510.
[37] L.K. Johannessen, G. Ellingsen, Integration and generification–agile software development in the healthcare market, Comput. Support. Coop. Work 18 (5–6) (2009) 607–634.
[38] L. Karlsson, T. Thelin, B. Regnell, P. Berander, C. Wohlin, Pair-wise comparisons versus planning game partitioning—experiments on requirements prioritisation techniques, Empir. Softw. Eng. 12 (1) (2007) 3–33.
[39] M. Khurum, Strategic Decision Support for Software Intensive Product Management, Licentiate Thesis, Blekinge Institute of Technology, Sweden, 2009.
[40] M. Khurum, K. Aslam, T. Gorschek, MERTS – A Method for Early Requirements Triage and Selection Utilizing Product Strategies, in: Proceedings of the Asian Pacific Software Engineering Conference (APSEC 07), 2007.
[41] A.J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M.B. Rosson, G. Rothermel, M. Shaw, S. Wiedenbeck, The state of the art in end-user software engineering, ACM Comput. Surv. (CSUR) 43 (3) (2011) 21, http://dx.doi.org/10.1145/1922649.1922658.
[42] M. Lehman, On understanding law, evolution, and conservation in the large-program life cycle, Syst. Softw. 1 (3) (1980) 213–231.
[43] H. Lieberman, F. Paternò, V. Wulf (Eds.), End-User Development (Human–Computer Interaction Series), vol. 9, Springer, Dordrecht, The Netherlands, 2006.
[44] B.P. Lientz, E.B. Swanson, Problems in application software maintenance, Commun. ACM 24 (11) (1981) 763–769.
[45] N.H. Madhavji, J. Fernández-Ramil, D.E. Perry (Eds.), Software Evolution and Feedback. Theory and Practice, John Wiley & Sons Ltd., 2006.
[46] T. Mens, S. Demeyer (Eds.), Software Evolution, Springer, 2008.
[47] E. Midleton-Kelly, IT legacy systems: enabling environment that reduce the legacy problem: a complexity perspective, in: N.H. Madhavji, J. Fernández-Ramil, D.E. Perry (Eds.), Software Evolution and Feedback. Theory and Practice, John Wiley & Sons Ltd., 2006.
[48] J. Pedersen, Protocols of Research and Design: Reflections on a Participatory Design Project (sort of), PhD Thesis, IT University of Copenhagen, Denmark, 2008.
[49] N. Pollock, R. Williams, L. D'Adderio, Global software and its provenance: generification work in the production of organizational software packages, Soc. Stud. Sci. 37 (2) (2007) 254–280.
[50] K. Rautiainen, Cycles of Control: A Temporal Pacing Framework for Software Product Development Management, Licentiate Thesis, Helsinki University of Technology, Finland, 2004.
[51] C. Robson, Real World Research: A Resource for Social Scientists and Practitioner-Researchers, second ed., Blackwell publishing, UK, 2002.
[52] K. Rönkkö, M. Hellman, Y, Dittrich, PD method and socio-political context of the development organisation, in: Participatory Design Conference, Indiana, US, October 1–4, ACM, 2008, pp. 71–80.
[53] K. Rönkkö, B. Kilander, M. Hellman, Y. Dittrich, Personas is not applicable: local remedies interpreted in a wider context, in: Proceedings of the Participatory Design Conference PDC, Toronto, July 27–31, ACM, 2004, pp. 112–120.
[54] W. Scacchi, T.A. Alspaugh, Understanding the role of licenses and evolution in open architecture software ecosystems, J. Syst. Softw. 85 (7) (2012) 1479–1494.
[55] K. Schwaber, M. Beedle, Agile Software Development with Scrum, first ed., Prentice Hall PTR, 2001.
[56] P. Sestoft, S. Vaucouleur, Technologies for evolvable software products: the conflict between customizations and evolution, in: Advances in Software Engineering. LNCS 5316, Springer-Verlag, 2008. http://dx.doi.org/10.1007/978-3-540-89762-0_8.
[57] C.R.B. de Souza, D.F. Redmiles, The awareness network, to whom should I display my actions? Whose actions should I monitor?, IEEE Trans Software Eng. 37 (2011) 3, http://dx.doi.org/10.1109/TSE.2011.19.
[58] A. Strauss, J.M. Corbin, Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, SAGE Publications, 1998.
[59] H. Unphon, Re-Engineering for Evolvability: Considering Social as well as Technical Requirements for Software Products, PhD Thesis, IT University of Copenhagen, Denmark, 2010.
[60] H. Unphon, Y. Dittrich, Organisation matters: how the organisation of software development influences the development of product line architecture, in: IASTED International Conference on Software Engineering, 2008, pp. 178–183.
[61] H. Unphon, Y. Dittrich, Architecture awareness in industrial software development, J. Syst. Softw. 83 (2010) 2211–2226.
[62] S. Vaucouleur, Software Customization by Code Query, PhD Thesis, IT University of Copenhagen, Denmark, 2009.
[63] I. van de Weerd, Advancing in Software Product Management: An Incremental Method Engineering Approach, PhD Thesis, Utrecht University, The Netherlands, 2009.
[64] B. Wessels, S. Walsh, E. Adam, Mediating voices: community participation in the design of E-enabled community care services, Inform. Soc. 24 (2008) 30–39.
[65] G. Zorn-Pauli, B. Paech, T. Beck, H. Karey, G. Ruhe, Analyzing an industrial strategic release planning process–a case study at Roche diagnostics, in: J. Doerr, A.L. Opdahl (Eds.), Requirements Engineering: Foundation for Software Quality, Springer, Berlin, Heidelberg, 2013, pp. 269–284. http://dx.doi.org/10.1007/978-3-642-37422-7_19.
[66] H. Züllighoven, Object-oriented Construction Handbook: Developing Application-oriented Software with the Tools & Materials Approach, Elsevier, 2004.