Taking Care of Cooperation when Evolving Socially Embedded Systems: The PloneMeeting Case

Hataichanok Unphon, Yvonne Dittrich
Software Development Group
IT University of Copenhagen
Denmark
{unphon, ydi}@itu.dk

Arnaud Hubaux
PReCISE Research Centre
Faculty of Computer Science
University of Namur
ahu@info.fundp.ac.be

Abstract

This paper proposes a framework to (i) analyse the contexts of socially embedded systems and (ii) support the understanding of change during their evolutions. Our finding is based on a co-operative project with a government agency developing a partially-automated variability configurator for an open source software product family. By employing our framework, we realised that the way variations and their management are implemented have to accommodate work practices from the use context as well as development practice, and here especially the cooperation within the development team and between users and developers. The empirical evidence has confirmed our understanding of what is relevant when estimating the evolvability of socially embedded systems. We propose to use our framework in architecture-level design and evaluation in order to take these cooperative relationships into account early in the evolution cycle.

1. Introduction

In this paper, we study the effects of change on the design of software from the socio-technical perspective. Our objectives are to (1) clarify the meaning of socially embedded systems, and (2) understand how to support their further development. More specifically, we are interested in understanding how developers can evaluate the impact of changes on the cooperation within the team and between developers and users. We report here on a study of an open source project developed by a Belgian government agency. The case study has confirmed our prior understanding of what dimensions of categories to investigate. A model of contextual categories relevant to assess design proposals developed in [21] is successfully applied to the case. The study confirms that the cooperation between developers and between developers and users have to be considered when evolving the architecture of a software product. The analysis using the framework provided us with a better understanding of the project and helped us to improve and complement the development process with state-of-the-art techniques. We therefore recommend using the model as a tool for structuring design discussions when drafting architecture and when evaluating evolutions for socially embedded systems.

This paper is outlined as follows. Section 2 introduces terms and definitions. Section 3 presents case description. Section 4 explains the research approach. Section 5 illustrates evolution of the PloneMeeting. Section 6 is discussion. Section 7 is conclusion and plan for future works.

2. Terms and definitions

This section defines two terms: (1) socially embedded systems, and (2) software evolvability, as shown in sections 2.1 and 2.2 respectively. To avoid terminological confusions between system and software, they are used interchangeably.

2.1. Socially embedded systems

An embedded system has ongoing interaction with a dynamic external world [15]. The concept of embedded systems is widely known in terms of a combination of computer hardware and software, and potentially additional mechanical or other parts, designed to perform a dedicated function. Wolf [22] defined the term embedded systems as 'any computer that is a component in a larger system and that relies on its own microprocessor'. In this paper, we call such systems *technically embedded systems*. Typical examples are MP3 players, telephone switches, and hybrid cars. Usually, design decisions are mainly constrained by interfaces to hardware or mechanical specifications. We define the term *socially embedded*

978-1-4244-3712-2/09/$25.00

systems as *any system that can be modelled intensively according to the environment and practices of its end-users*. ERP systems, e-government applications, virtual office software, and decision support systems are examples of socially embedded systems. Design decisions of socially embedded systems underline the importance of the human interaction with and cooperation via the systems for the societal activities. Lehman [18] has defined a characteristic of *E*mbedded programs (*E*-programs) that it has become a part of the world which it models. This implies a constant pressure for change. Since focused human or societal activities, the usability of the system is the main concern of *E*-programs. The close co-operation between end-users, people working with the systems on a daily basis, and developers throughout the entire development process is strongly recommended for capturing the contexts and qualities of use that cannot be fully anticipated at the initial phase.

Participatory Design (PD) is a research discourse investigating in how to support cooperation between users and developers when designing socially embedded systems [17]. The focus of *PD* is to let IT-designers work directly with their end-users in the end-user's own environment and come up with design ideas in real-life work situations. Floyd et al. [12] have proposed *STEPS*, *S*oftware *T*echnology for *E*volutionary *P*articipative *S*ystem Development (*STEPS*) in order to promote user-developer cooperation as the use context is considered an inherent a part of the development. In this approach, software development does not start from pre-defined problems, but must be considered as a learning process involving the unfolding of the problems as well as the elaboration of a solution tackling the problems. Dittrich et al. [9] elaborated on how co-operation with the users can take place in an industrial project without jeopardising the planning and control of the development process. By employing mock-ups, prototypes, or scenarios of use, end-users can experience the new technology, and IT-designers can experience the new work practice.

Socially embedded systems often allow users to tailor the software to specific needs. Examples of end-user tailoring categories are customisation, composition, expansion, and extension [11]. Apart from tailoring, the socially embedded systems also have to evolve over time.

2.2. Software evolvability

Belady and Lehman [3] first introduced and used the term *evolution* as 'a sequence of changes to the system over its lifetime which encompassed both development and maintenance'. Cook et al. [4] has developed the concept of software evolvability based on maintainability characteristics in ISO 9126, and proposed the evolvability measurement at different levels of abstraction, i.e., at pre-design, architectural, detailed design and source code levels. The concept of evolvability brings together factors from three main areas: (1) software product quality, (2) software evolution processes, and (3) the organizational environment in which the software is used. They have defined the concept of software evolvability as 'the capability of software products to be evolved to continue to serve their customer in a cost-effective way'.

To survive in today's competitive software market, it would be too restrictive to limit software evolvability to maintenance issues only. The growth dynamics of a system depends highly on the business context. To increase a market share, e.g., it may be vital to bring out new features. Yet, a system that is used will be changed [19]. We here define *software evolvability* as *the adaptability of software in order to serve the needs of use and business contexts over time reflecting on its architecture*. Software architecture represents a common abstraction of a system that many of the system's stakeholders can use as a basis for mutual understanding, negotiation, consensus, and communication [2]. When the needs of use and business contexts trigger changes to the software, the stakeholders must handle the needs based on the architecture.

In this paper, we address how to relate cooperation among developers and between users and developers when evolving socially embedded systems. Based on previous research [21], we recommend framework that builds up on six contexts: business, use, software engineering organisation, software engineering practice, technical infrastructure, and technical selection. The *business context* is the context or environment to which the system belongs. The *use context* relates the system to the work practices of the intended users. The *software engineering organisation* is the organisational context in which the software development is carried out. The *software engineering practice* refers to the analysis of the work practices of the system developers. The *technical infrastructure* lists the hardware and basic software assets backing the system. The *technical selection* is part of a suggested design and needs to be seen in the context of existing and planned systems, as well as in the context of other systems that are part of the same design.

Others have used the notion of context or contextual factors before. Kensing [16] has proposed a conceptual framework that IT-designers should be aware of when they design IT-applications to meet the needs of a specific organisation. The framework addresses: (1) project context, separating into design context and implementation context; (2) use context, dealing with

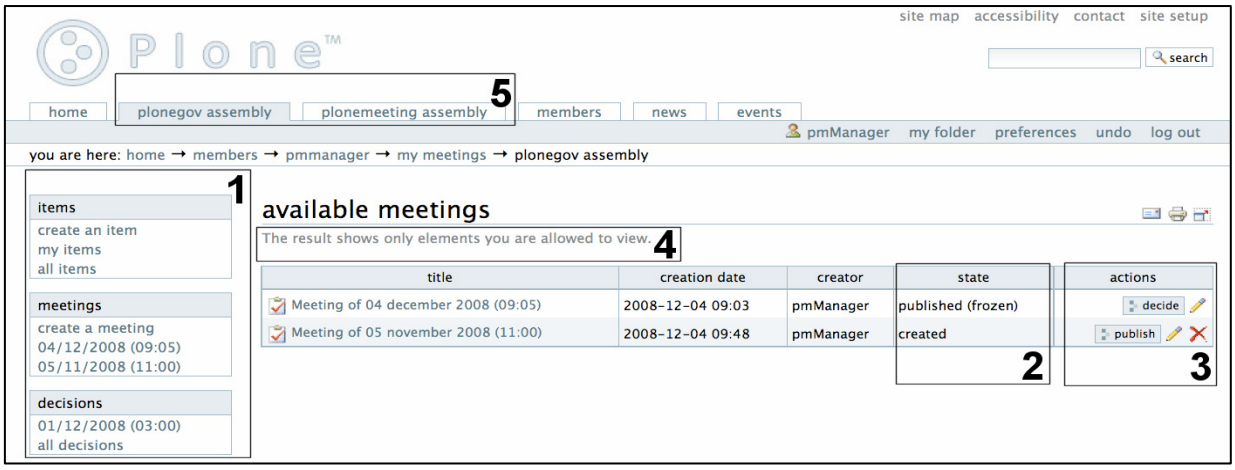

Figure 1. The PloneMeeting graphic user interface

work practice context and strategic context; and (3) technical context, interacting with system context and platform context. Kensing does not apply the framework to reflect on concrete design proposals. Dittrich and Lindeberg [8] developed Kensing's framework further by mapping out contextual factors in order to understand the suitability of a – from a technical viewpoint – less advanced design for a specific industrial setting. Here, three contextual dimensions are used: development, use, and technical. We developed this framework to support architecture-based analysis when planning to evolve software products.

3. Case description

EASI-WAL[1] is the Belgian government agency in charge of (1) the simplification of administrative tasks and (2) the global computerisation of the administrative processes of the Walloon region[2]. To this end, EASI-WAL develops open source web-based applications assisting the Walloon public institutions. In order to stay independent of external contractors and to pool the efforts, the developers set out to develop applications generic enough to deal with the difference in scales and behavioural disparities of the various institutions.

For each of their products, the developers struggled to properly elicit and express the requirements. Although, the elicitation of the requirements followed an iterative process of prototype demonstration and validation with users together with impromptu interviews with the users and results of developers' experience in the field, it did not follow a specific elicitation protocol. As we examined their case, it turned out that most of the challenges came from the identification of the common parts, shared by all the institutions, and the specific parts, peculiar to a single or a set of them. We thus advocated software product line engineering [20] as solution to their problem of systematising the engineering of product families. In order to assess the potential of software product line engineering, the developers agreed to apply it to an existing application, *viz*. PloneMeeting [7, 14].

PloneMeeting provides advanced meeting management functionalities like meeting workflow specifications and document generation. Figure 1 shows a screenshot of the graphical user interface of PloneMeeting. To further illustrate the functionalities proposed by PloneMeeting, we trace three representative use cases: meeting management, meeting workflow management, and document generation.

Meeting management usually follows a three-step process. First, the meeting items are created and validated. Secondly, a meeting is created and existing meeting items are added to the meeting agenda. Third, after publication, the meeting takes place and the decisions related to the meeting items are recorded (label 1 in Figure 1, a.k.a. Figure1.1).

Meeting workflow management. Central to PloneMeeting is the concept of meeting itself. Each meeting is associated to a state of a workflow. The meeting states evolve according to a pre-defined workflow designed and selected at the installation of PloneMeeting. A typical workflow contains states like *Created*, *Published, Frozen, Decided*, *Closed and Archived* (Figure 1.2). The respective next states can be seen in the *actions* column (Figure 1.3). On top of specifying valid states, a workflow contains guards and potential actions on the transitions. Guards basically check the permissions of the user willing to trigger the transition (Figure 1.4). The available workflows are

[1] The EASI-WAL website, http://easi.wallonie.be/.

[2] The Walloon region gathers a third of the Belgian population and covers about a half of the territory. The region includes a French-speaking and a minor German-speaking community.

selected during the PloneMeeting base configuration. The user selects one of these workflows for each meeting created. The portal tabs show two meeting configurations, i.e. *plonegov assembly* and *plonemeeting assembly* (Figure 1.5).

Document generation. Document generation is an essential functionality that must be provided by a meeting management system dedicated to governmental administration. For that reason, every meeting item, meeting and decision can be exported into various formats like PDF, ODT and DOC in a single click. Different document templates can be specified and selected in the PloneMeeting configuration menu.

4. Research approach

Our research approach is a combination of co-operative method development (CMD) [10] and grounded theory approach [13]. The CMD is a domain-specific adaptation of action research consisting of three evolutional phases: (1) understanding practice, (2) deliberate improvements, and (3) implement and observe improvements. The grounded theory approach is an explicit and systematic technique for developing theory iteratively from qualitative data. The initial analysis of data begins without any preconceived categories. When an interesting pattern emerges, it is repeatedly compared with existing data, and additional data is collected to support or refute the emerging theory. We have applied the grounded theory approach to our observation and field notes, transcripts of interviews and workshops, and other material collected through these activities including relevant literatures.

We have applied the CMD to the basis of our fieldwork research aiming at institutionalising variability management for the PloneMeeting product family. Initially, we conducted interviews with three PloneMeeting developers, one working for EASI-WAL and the two others working for two different city councils. We studied functionality of the existing application, structure of the source code, and checked out the major related product, i.e. PloneTask. Besides these contact meetings, we held two workshops in which we successively studied: (1) the product line architecture of PloneMeeting, and (2) the feature modelling and configuration tools. In the first workshop, the main responsible developer of PloneMeeting reflected and explained the current architecture in terms of implementation techniques and rationales behind it. In the second workshop, we presented a number of mainstream variability management tools on the market and suggested a work plan for integrating an automated configuration tool into the existing PloneMeeting architecture. The development of the configurator is currently supervised by a group of researchers from the PReCISE research centre[3], University of Namur, Belgium.

5. Evolution of PloneMeeting

This section reveals the evolutionary story of PloneMeeting into three episodes: (1) Collège: the origin of PloneMeeting, (2) the PloneMeeting product family today, and (3) towards the PloneMeeting configuration wizards, as shown in sections 5.1, 5.2, and 5.3 respectively. For each of these episodes, we will look into the contexts presented in section 2.2.

5.1. Collège: the origin of PloneMeeting

Business context. Collège was a web-based application dedicated to the management of the official meeting of municipal authorities in Belgium. Developed in 2004, Collège has been in production three years. Part of the Belgium policy for e-government is to use open source software as much as possible.

Technical infrastructure. Collège is a monolithic program or a self-contained program[4].

The **technical selection** is not available.

Software engineering practice. The developers report that the evolution of Collège was a challenge. Due to a lack of explicit architecture, most modifications were done at the source code level. Implementing a simple meeting template took a day. Templates could not be shared easily.

Software engineering organisation. The Collège developer team was located in the same region. When a city requested a Collège product, the team copied the existing Collège source code and modified it according to the city's specific needs. In case of updates, a power-user, who acted as administrator, had to go through the whole code base and manually update the code and resolve the conflicts.

Use context. Laws and legal regulations constrained use cases. For example, Collège can generate a document only in PDF format because an official document should not be modified.

It seemed impossible for users to change anything apart from three or four configurable elements. Most of the changes involved programming. During the first workshop on the PloneMeeting architecture, a developer mentioned that "*Collège was used in three cities. …Three cities are OK, but what if you have 45 cities? This way of doing things is not manageable. This product was not designed correctly for a lot of*

[3] The PReCISE website, http://www.fundp.ac.be/universite/interfacultaire/precise/.

[4] Monolithic program or self-contained program indicates a program which is contained in a single function of the large program.

organisations. That is the first limitation of the old architecture. ".

5.2. The PloneMeeting product family today

Business context. Since June 2007, PloneMeeting has been totally refactored from Collège and has gathered interest from other government agencies. PloneMeeting has been tailored for four main organisation types, i.e. general purpose, city, government, and parliament. A version for the parliament has been planned.

Use context. PloneMeeting is used by civil servants. The amount of users per product is approximately 100 with up to 10 different roles like meeting manager or reviewer. The products are customised. Changing the display language and document layout are examples for simple customisations.

When installing PloneMeeting power-users also can select their own workflows based on pre-defined UML state charts available in the PloneMeeting set of core assets. A power-user is free to implement its own workflow but it requires advanced programming skills.

Technical infrastructure. PloneMeeting is developed on the top of an open source technology stack: Plone[5], Zope[6], and Python[7].

Technical selection. Since PloneMeeting is built on top of Plone, some architectural dependencies are directly managed by Plone. Using a plug-in architecture, the core of PloneMeeting is relatively simple and flexible enough so that nearly every aspect of its input and output can be modified by the plug-in. This mechanism is used for customising the behaviours of PloneMeeting.

The most sensible part of the PloneMeeting architecture is the workflow. A workflow is composed of several states, actions, and guards and involves several categories of users. PloneMeeting comes with several workflows so as to meet the work practices of most organisations. PloneMeeting employs ArchGenXML to generate Python code from the workflows designed in UML before the product configuration. Although ArchGenXML builds on architecture-centric, model-based and test-driven development, PloneMeeting developers still have to write the code e.g. the body of a method or the trigger of a workflow transition manually.

The current configuration tools do not enforce any constraints, e.g. a power-user can choose more than one option for a feature even if they exclude each other.

Software engineering organisation. PloneMeeting belongs to larger open source project, namely PloneGov[8]. Five developers belonging to different municipalities or organisations are working part time on PloneMeeting. The developers gather requirements from their own organisations/entities. The architectural change occurs after developers get feedback from end-users.

One of the developers has the main and coordinating responsibility. He reviews the modifications, checks in the updated source code, and tests it. To develop a subsequent product, every change is done after a careful discussion with all the developers.

PloneMeeting developers do not only provide the software products to open source communities, but also are power-users and administrators for their own organisations. To reduce development time, the vision is to simplify configuration and leave it to power-users who have no development charges. When we discussed the configuration of the product, we were puzzled by the fact that configuration choices can be performed at different times in the lifecycle. This absence of control would notably allow configurations incompatible with the current state of the system (e.g. unknown workflow states for running meeting objects). We thus started to differentiate configuration according to the binding time and evolvability of the choices – some can only be done once when installing the software, others can be changed repeatedly as part of the usage. Still, categorisation needs to be clarified in a systematic way. One approach would be to employ the categories used in end-user tailoring: customisation, composition, expansion, and extension, instead of the term configuration.

The **software engineering practice** looks different depending on the role in the development process. We interviewed a developer working for an organisation that wants to replace their old systems with PloneMeeting. In the beginning, the developer met end-users of the old systems every second week. The developer presented a prototype of PloneMeeting and the end-users commented on it. He recorded the requirements and change requests to the Trac System, an enhanced Wiki and issue tracking system. Then he changed the configurations, implemented and released the necessary code changes, which took about 2 weeks.

PloneMeeting developers join in weekly *sprints*. A sprint is a focused development session lasting anywhere from a day to a week. During sprints, the developers prioritise the requirements, design, code, test and at the end of a day release a new version of PloneMeeting. Criteria for giving high priority

[5] The Plone website, http://plone.org.
[6] The Zope website, http://www.zope.org.
[7] The Python website, http://www.python.org.
[8] The PloneGov website, http://www.plonegov.org

requirements are (1) a requirement is highly required from several organisations, and (2) a requirement comes from the organisation that allocates the most resources, e.g. time, to the project. Most of the requirements are shared among organisations.

Developers are located very close to the user organisations for which they create the local installation. For this, they have to configure and adapt a PloneMeeting profile manually, which is a time consuming task considering the great deal of technologies involved. To generate a skeleton of a product, they reuse and adapt, if needs be, the existing UML models.

As much of the development is model driven, most of the code is generated. The architecture is not explicitly documented. The parallel development provides a challenge for the model driven approach as changes have to be coordinated at the model level as well as at the source code level. Since the deployment is dynamic, modification of the core product are automatically deployed on every new installation of PloneMeeting. In case of major changes that affect the data structures, the developers provide a script for data migration so that local versions can be upgraded.

Another time-consuming task for developers is to handle change requests after base installation. An example is the archive meeting case. A stakeholder requested to archive documentation of meetings. In order to add this unanticipated change request, the developer needed to change a number of parameters, customise and add some status to the workflow, write some Python method, etc. From the developers' point of view, a critical issue in the shared development environment is an architecture that allows developers to capture commonality of the organisations into common parts as well as to easily integrate specificity among organisations that share and use PloneMeeting.

5.3. Towards the PloneMeeting wizards

A practical outcome of our research co-operation and the facts revealed with our framework is the development of an automated configurator for PloneMeeting. During the second workshop on feature modelling and configuration tools, we came up with a plan for the PloneMeeting wizard and configuration tool. Table 1 summarised the ideas and will be detailed later in this section. The difficulties described above indicate the need to better structure the configuration tasks. Different actors who configure and customise different aspects of the software at different points in time need different kinds of support. From the introduction of the development, business context and software engineering organisation, and technical infrastructure remain the same, but the use context, software engineering practice, and technology selection are impacted.

Table 1. Plan for PloneMeeting wizards and configuration tools

KIND OF CONFIG.	Initial install	Product line related change	Simple parameter change
FREQ. OF USE	Once	Once a year	Once a week
ACTOR	Power-user with business knowledge	Power-user with business knowledge	Power-user with technical knowledge
INPUT/ PRE-COND.	• Business analysis	• Business analysis • PloneMeeting plug-in to Plone	Simple parameter change request
OUTPUT / POST-COND.	• PloneMeeting plug-in to Plone • BuildOut	• Adapted PloneMeeting plug-in • Adapted BuildOut • Migration script (if big change)	Configuration data change
COMPONENT TO USE	Wizard	Wizard	Configuration tool

Use context. PloneMeeting has at least 15 power-users from different government agencies in Belgium. PloneMeeting developers have an ambition to support their power-users to configure bigger parts of the software and be as independent from developers as possible. Although the power-users have little knowledge of software engineering, the wizards and the configuration tools can enable them to express their requirements by choosing from a list of existing features. For instance, to implement a meeting archive feature, they would use a wizard. The wizard would generate an instance of PloneMeeting product family using a given workflow including the meeting archive.

However, different levels and frequencies of configuration have to be distinguished. When first installing the software, a power-user with business knowledge will configure the basic family member using one wizard. The wizard will generate some components. The subsequent stages represent modification of the base configuration. The tools supporting the latter stages constrain configurations to what is allowed with respect to the running configuration.

Software engineering practice. In order to create a wizard, developers are eliciting the user-level variability that will be used for developing the PloneMeeting wizard and configuration tool. This elicitation will be done at the requirements level. However, the requirements are not completely documented. Often, requirements gathering have been informal and chaotic. PloneMeeting has been refactored from Collège whose developer did neither use the Trac sys-

tem nor any other tool for documentation. Furthermore, the documentation of requirements and product analysis are not standardised. PloneMeeting developers now have an ambition to have robust processes for requirement elicitation and traceability. As a starting point for the development of the PloneMeeting wizard and the configuration tool, we suggested to list all the questions that end-users, power-users and developers should be asked in parallel with likely answers, in order to systemise their elicitation process.

Technical selection. Developers have distinguished configuration elements into features and simpler parameters. Email of power-user and meeting type are examples of a simple parameter. Document generation, document template, meeting archive and meeting workflow are examples of features. The simple parameter is used in settings that neither influence any feature nor effect on the rest of the configuration. The features are categorised into features that can be changed during runtime, e.g. the interface language, and features that are decided when first installing the product, e.g. the meeting workflow. If features need to be changed, the configured PloneMeeting instance must be removed and replaced by an updated version. There are thus three levels of configuration, which can change at different frequencies and require different expertises.

The currently used feature modelling language is the cardinality-based feature diagram of Czarnecki et al. [6]. The configuration process under evaluation is the multi-level staged configuration process, which offers a fine-grained and well-defined decomposition of the modelling perspectives into linked feature diagrams organised in sequential levels. The sustainability and gain of this approach are still to be diligently demonstrated.

In order to automate the build process, developers choose BuildOut, a tool for developing, packaging, and deploying Plone applications. Another design option that will facilitate maintenance would be to enhance the existing PloneMeeting configurator with automation and export/import facilities. At the time of writing, the PloneMeeting developers have decided to represent the feature diagram directly in Python in order to avoid any dependency to existing tools and ease the binding with the code.

6. Discussion

In the discussion we take up three points: (1) the use of the six contextual dimensions as a framework to prepare and evaluate changes to software products, (2) how to introduce the framework into the architecture evaluation practice in a company, and (3) how product line and product line configuration approaches that are developed for technically embedded systems can be applied to socially embedded systems.

Using the framework to design and evaluation of changes. The PloneMeeting case suggests that the six contextual dimensions – business, use, software engineering organisation, software engineering practice, technical infrastructure, and technical selection – can be applied in order to understand the effects of change for the socio-technical context of socially embedded systems. The six contexts helped us to map out aspects affected by changes which have to be considered and resolved when analysing specific requirements.

The analysis provided in this article helped to clarify the complexity of the configuration task for the PloneMeeting product family. When evaluating the concrete design for the configuration mechanism, the framework can be used to analyse and evaluate the impact of the chosen solution on the cooperation, respectively, the distribution of tasks between the local developers, the power-users, and end-users.

Introducing the framework into architecture evaluation practices. The introduction of a method into an existing software development organisation is not always a straightforward task. Ali Babar et al. [1] have presented an empirical investigation of factors influencing industrial architecture evaluation practices. They have categorised the findings into: (1) organisational factors, involving: engagement models, governance frameworks, supporting software engineering organisational structures, design decision documentations, funding models, and training; (2) technical factors, including: quality attributes being evaluated, challenges caused by integration issues, techniques and tools for representing and visualising architectures, types of evaluation required, and methods and guidelines used; (3) socio-political factors, relating to: soft skills, organisational politics, and vendor involvement; (4) managerial factors: pulling by management, support and commitment, objectives of evaluating architectures, and stakeholder-centric issues; and (5) business factors: resulting from business needs and industry standards, and requirements of business case. The CMD approach is based on cooperation between researchers and practitioners when deliberating, introducing and evaluating improvements and will thus provide a good base to explore the specific benefits and hinders when introducing our framework. So far, we did not explicitly introduce the framework into the development lifecycle. The analysis of its result helped us to better understand the context of PloneMeeting and worked as a catalyst to improve the development practices. This paper is only the first stage towards its understanding and uptake by practitioners. Additional

work is still needed to assess the actual impact it will have once used by the different stakeholders.

How to apply product-line architecture approaches to socially embedded systems? In the existing work, feature modelling has been applied to technically embedded systems [5]. But PloneMeeting is considered as a socially embedded system because of the extensive interactions with the environment and practices with users. The design decisions are loosely constrained by static conditions, and the contexts and qualities of use cannot be fully anticipated in the starting phase. The next step is to explore how feature modelling can be applied, in general, to socially embedded systems and evaluate to what extent it enhances their usability.

7. Conclusion and future works

The PloneMeeting case demonstrates how our framework can be used to understand the impact of changes in socially embedded systems on the cooperation among developers and between users and developers. In the PloneMeeting case, introducing a wizard and a configuration tool turned to solve some problems in the local development by controlling the instantiation of PloneMeeting and its features. From the early phase of the development of the wizard and the configuration tool, the changes are designed to support the work and development practices of the power-users.

At the time of writing, we are assessing the impact of the wizard and configuration tool on the development and deployment of the application in production sites. We are polishing up the terms and definitions as well as the analytical tool proposed in Section 2 with different cases. Apart from that, we keep up with the questions and challenges posed in the discussion section.

References

[1] M. Ali Babar, L. Bass, and I. Gorton, "Factors Influencing Industrial Practices of Software Architecture Evaluation: An Empirical Investigation", *in S. Overhage et al. (Eds.)*, *the 3rd Int. Conf. on the Quality of Software Architecture (QoSA)*, *LNCS 4880*, Spinger-Verlag, 2007, pp. 90-107.

[2] Bass, L., P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd edition, Addison-Wesley, 2003.

[3] L.A. Belady, and M.M. Lehman, "A Model of Large Program Development", *IBM Systems Journal 15(1)*, 1976, pp. 225-252.

[4] S. Cook, H. Ji, and R. Harrison, "Software Evolution and Software Evolvability", *Working paper*, University of Reading, UK, 2000.

[5] K. Czarnecki, T. Bednasch, P. Unger, and U. W. Eisenecker, "Generative Programming for Embedded Software: An Industrial Experience Report", *Proc. ACM SIGPLAN/SIGSOFT Conf. on Generative Programming and Component Engineering (GPCE'02)*, *LNCS 2487*, Springer-Verlag, Germany, 2002, pp. 156-172.

[6] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged Configuration Through Specialization and Multi-level Configuration of Feature Models", *Software Process: Improvement and Practice 10(2)*, *Special issue on Software Product Lines*, John Wiley & Sons, 2005, pp. 143-169.

[7] G. Delannay, K. Mens, K., P. Heymans, P.-Y. Schobbens, and J.-M. Zeippen, "PloneGov as an Open Source Product Line", *Proc. 3rd Int. Workshops on Open Source Software and Product Lines*, 2007.

[8] Dittrich, Y., and O. Lindeberg, Designing for changing work and business practices, *in Adaptive evolutionary information systems*, IGI Publishing, USA, 2003, pp. 152-171.

[9] Y. Dittrich, and O. Lindeberg, "How Use–Oriented Development can Take Place", *Information and Software Technology 46(9)*, 1 July 2004, pp. 603-617.

[10] Y. Dittrich, K. Rönkkö, J. Erikson, C. Hansson and O. Lindeberg, "Co-Operative Method Development: Combining qualitative empirical research with method, technical and process improvement", *Empirical Software Engineering Journal 13(3)*, Kluwer Academic Publishers, 2008, pp. 231-260.

[11] Eriksson, J., *Supporting the Cooperative Design Process of End-User Tailoring*, Doctoral Dissertation, Department of Interaction and System Design, School of Engineering, Blekinge Institute of Technology, Sweden, 2008.

[12] C. Floyd, F.-M. Reisin, and G. Schmidt, "STEPS to Software Development with Users", *Proc. 2nd European Software Engineering Conf.*, *LNCS 387*, 1989, pp. 48-64.

[13] Glaser, B.G., and A. Strauss, *Discovering of Grounded Theory: Strategies for Qualitative Research*, Sociology Press, 1967.

[14] A. Hubaux, P. Heymans, and H. Unphon, "Separating Variability Concerns in a Product Line Re-Engineering Project", *Proc. 2008 AOSD workshop on Early Aspects*, Brussels, Belgium, 2008.

[15] Kaelbling, L.P., *Learning in Embedded Systems*, MIT Press, 1993.

[16] F. Kensing, "Participatory Design in a Commercial Context – a Conceptual Framework", *Proc. Participatory Design Conf.*, USA, 28 Nov.-1 Dec. 2000, pp. 116-126.

[17] Kensing, F., *Methods and Practices in Participatory Design*, ITU Press, Denmark, 2003.

[18] M.M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", *IEEE 68(9)*, 1980, pp. 1060-1076.

[19] M.M. Lehman, "On Understanding Law, Evolution, and Conservation in the Large-Program Life Cycle", *Systems and Software 1(3)*, 1980, pp. 213-231.

[20] Pohl, K., G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer-Verlag, USA, 2005.

[21] H. Unphon, and Y. Dittrich, "Organisation matters: How the Organisation of Software Development Influences the Development of Product Line Architecture", *Proc. IASTED Int. Conf. on Software Engineering*, Innsbruck, Austria, 2008 , pp. 178-183.

[22] W. Wolf, "What is Embedded Computing?", *Computer 35(1)*, IEEE Computer Society, 2002, pp. 136-137.