Please quote as: Geihs, K.; Leimeister, J. M.; Roßnagel, A. & Schmidt, L. (2012): On Socio-technical Enablers for Ubiquitous Computing Applications. In: 3rd Workshop on Enablers for Ubiquitous Computing and Smart Services (EUCASS 2012), at 2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet (SAINT), Izmir, Turkey.

On Socio-technical Enablers for Ubiquitous Computing Applications

Kurt Geihs, Jan Marco Leimeister, Alexander Roßnagel, Ludger Schmidt
University of Kassel
D-34121 Kassel, Germany
[geihs, leimeister, rossnagel, l.schmidt]@uni-kassel.de

***Abstract* —** The focus of this paper is on context-aware, self-adaptive ubiquitous computing applications that involve mobile users. The development of such applications is inherently complex for two main reasons: From a technical perspective, context management and adaptation management add complexity to the application design and implementation. From a socio-technical perspective, concerns and requirements related to the social embedding and user acceptance must be addressed in the application design and lead to additional complexity, particularly because sensitive personal data is collected, processed, stored and communicated by such applications. In this position paper we present an analysis of the problem space and a solution approach. We have developed an interdisciplinary methodology that systematically addresses technical as well as non-technical concerns. Our conclusion is that solving the socio-technical challenges will be a key enabler for ubiquitous computing.

Keywords: ubiquitous computing, application development, methodology, socio-technical aspects

I. INTRODUCTION

Ubiquitous computing (UC) is an exciting paradigm shift that embraces a model in which computing resources and services blend seamlessly with our daily life environment and are discovered and bound dynamically at run-time. Typically, UC applications involve mobile users using applications that are context-aware and self-adaptive, i.e. applications self-adapt during run-time to their changing context in order to maintain or improve their functionality and quality of service. This creates an enormous potential for innovative applications that intelligently support the user in reaction to her current situation.

However, there is a flip side of the coin: the development of such applications is inherently complex. This is due to two main reasons. From a technical point of view, not only need the developers understand the main functionality of the application and how this can be provided on a mobile device, but also they have to conceive different application variants, specify how application variants are linked to the execution context state, and determine which variant should be activated under which context conditions. This complexity may easily appear like an insurmountable barrier to the developer if appropriate software development and run-time support is missing. Lately, systematic software engineering support has been made available for the development of context-aware and self-adaptive mobile applications. One example for such a development framework was delivered by the European project MUSIC (Self-Adapting Applications for Mobile Users in Ubiquitous Computing Environments) [10].

The second challenge in the development of context-aware, adaptive, mobile ubiquitous computing applications arises especially if the objective is not just a technically sound application but a system that is widely accepted by users. This challenge is due to the user-centric nature of such applications, i.e. the socio-technical dimension. It comprises aspects such as privacy, trust, usability, and legal considerations and has not been addressed systematically in development methodologies for UC applications so far. The socio-technical aspects are concerned about the user as the focal point of the processing activities. For UC applications questions arise immediately about the acceptance and social embedding of the new technology. We claim that UC applications in particular require careful consideration of user-related socio-technical aspects. These aspects must be an integral part of the software development process. Addressing and solving these challenges will be a key enabler for UC. Otherwise there will be a lack of acceptance of such a user-centric technology.

In this position paper we present an analysis of the problem space as well as an approach to tackle the challenges. We report on the results of an ongoing collaborative, interdisciplinary research project where we are developing a development methodology that takes technical as well as non-technical requirements and concerns into account in a systematic and integrated fashion.

In Section II we discuss the requirements for a comprehensive development methodology for UC systems. Section III presents our interdisciplinary development approach. In Section IV we discuss related work. Section V presents preliminary experiences with the new methodology and conclusions.

II. REQUIREMENTS

Widespread adoption of a new technology, in particular if it is a user-centric technology such as context-aware, adaptive ubiquitous computing, not only depends on the technical progress, but also on "soft factors" that determine the user acceptance. Our goal is to develop a software engineering method that encompasses the socio-technical aspects of application systems as first class requirements and facilitates the development of applications that are socially compatible by design. We intend to avoid the often encountered situation that a software product is rejected because it has non-technical flaws and risks.

Essentially, the utility of a system is determined by both its functional and non-functional qualities. Both quality dimensions must be taken into account in the development of software. However, the software engineering community agrees that existing software engineering methodologies focus essentially on *"notations and techniques for defining and providing the functions a software system has to perform"* [1]. Furthermore, there is a large variety of definitions

and characterizations of the term *non-functional requirement* as well as a rather large number of classification schemes and taxonomies for such requirements. ([1] presents an excellent overview.) Typical examples for non-functional concerns in the design of software systems are: performance, throughput, reliability, robustness, portability, testability, maintainability, usability, and many more.

In our work we have asked ourselves what non-functional requirements are crucial to the acceptance, i.e. social embedding, of user-centered context-aware UC applications, and we decided to concentrate first on three key concerns: trust, usability, and legal conformance. In order to emphasize that we focus particularly on software qualities that are related to the social embedding of the software, we call these concerns and requirements *socio-technical*.

Certainly, security and privacy of user data are very important concerns in UC applications. From our perspective on UC, we view these elementary concerns as part of the technical requirements that need to be designed into the technical solution and cannot be added later on. Therefore, in the following text we implicitly include security and privacy in the set of technical requirements of an application. Note that specific security and privacy mechanisms may be employed to provide technical support for socio-technical requirements related to trust or legal conformance.

Thus, our development methodology addresses particularly the following socio-technical concerns and questions:

- **Trust:**
  How can the user build up trust in a system which monitors the user's context and adapts automatically? Does the system really behave as the user wants it to behave? What kind of technical mechanisms support trust-building of users? How and where are trust-supporting components integrated into UC systems?
- **Usability:**
  How can we make sure that the user can handle and interact with a system where many components are hidden in the environment and where many activities happen automatically? How does the user react to partially losing control when using an UC application?
- **Conformance to legal regulations:**
  How can we include legal considerations into the design process such that the processing, storing and sending of application data do not violate existing law? What kind of service contracts do we use (implicitly or explicitly) if third party service providers are involved?

Clearly, we could have included (and we will include in future work) more than exactly these three socio-technical aspects in our methodology, such as motivational and business concerns. However, for practical reasons we decided to focus first on these three. We believe that the chosen aspects represent a fairly diverse and broad spectrum of socio-technical requirements.

Many different software engineering methods are available today. There is no "one size fits all" method. Ideally, the integration of socio-technical aspects into a software engineering method should be agnostic to the specific kind of software engineering methodology used. Whether a classic spiral development approach or Scrum is used, should not influence the degree of social compatibility of the finished product.

III. Development Approach

Efficient, effective and high quality software development is becoming increasingly important in today's world. However, socio-technical requirements are difficult to assess. A major part of the development effort is spent for the requirements analysis and the transformation of socio-technical requirements into technical artifacts.

Our methodology for the development of UC applications covers the conventional software development phases requirements analysis, conceptual design, software design, implementation, and evaluation. These phases may be walked through in several iterations. The socio-technical concerns trust, usability and legal conformance are discussed and monitored in all phases, but most effort in respect to dealing with these requirements is spent in the requirements analysis and conceptual design phases. In the following we focus specifically on how the socio-technical concerns are addressed.

It goes almost without saying that the development team of an UC application must be an interdisciplinary team, i.e. must consist of requirements engineers and software engineers as well as domain experts for the socio-technical disciplines.

A. Requirements Analysis

Usability is defined as *"the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use"* [2]. Usability is observable only at runtime. It is a major concern with interactive systems. For acquiring usability requirements the usage context is to be analyzed. Components to be considered are users, tasks, equipment, and the physical and social environments. By definition, the user is the most important one. Thus, usability engineering involves the user either by having the intended users of a product in mind (i.e. user-centered design) or by making users actual members of the design team (i.e. participatory design).

For UC applications we favour participatory design. The users should be involved from the start in order to validate potential usage scenarios and user roles. This leads to a set of usability requirements that become part of the overall application requirements.

Research on technology acceptance has shown that **trust** is a key determinant of technology adoption [7]. Our notion of trust is based on the definition of [8] and defined as the belief *"that an agent will help achieve an individual's goal in a situation characterized by uncertainty and vulnerability"*.

This leads to three concerns that we consider most important for adaptive UC applications: understandability, i.e. covering the aspect of how good the user was able to understand how the system works, control, i.e. dealing with the

degree to which the user has the feeling of having the system under control, and information accuracy, i.e. focusing on the aspect that the information provided by the system is accurate. Each of these three concerns leads to requirements that are prioritized according to importance and added to the set of requirements.

From a **legal** perspective, we face the difficulty that legal norms typically are farthest away from obvious mappings into technical artefacts. They rarely contain concrete provisions for the design of technical systems. Nevertheless, developers of UC applications must make sure that legal provisions are not violated.

To acquire technical requirements from legal provisions, our methodology builds on a method called KORA [9]. KORA specifies a four-step refinement process of how legal provisions get concretized step-by-step to technical requirements.

Towards the end of the requirements analysis phase the functional application requirements, derived by conventional requirements engineering, need to be examined together with the socio-technical requirements related to trust, usability, and legal conformance. There may be conflicting requirements, requirements that incur too much effort or cost, etc. Thus, the final requirements negotiation is an important step of the requirements analysis phase which may lead to deleting, adding or reworking some of the requirements. Finally, all requirements should be expressed in a way that the developers can understand them.

B. Conceptual design

Based on the derived set of requirements the development team jointly builds a consistent conceptual model of the application. This is performed in six steps.

The first step is to review the requirements that resulted from the previous phase. This results in a mapping of one or more requirements onto technical features that the application should provide.

In the second step fine-grained use-cases are defined by requirements and software engineers in order to concretize the targeted application functionality. The use cases are reviewed by the experts for trust engineering, law, and usability rom the perspectives of their disciplines. This leads to further consolidation and refinement of requirements.

The third step produces the structure of the user interface by iteratively creating flow charts and a sitemap. Flow charts visualize the usage of the application, usually centred on a specific task or function. While a flow chart consists of a series of screens that collect and display information to the users, a sitemap illustrates the hierarchy of screens.

Based on the use cases, flow charts, and sitemap the user interface can be designed in a fourth step. This involves positioning and fine-tuning the content in each view (in so-called wireframes) and the development of an overall visual screen design. Specific system design guidelines of the target platform must be taken into account. Note that while steps three and four are performed mainly by software and usability engineers, experts from the other two socio-technical dimensions constantly evaluate the results and may contribute their opinions and requirements.

In step five trust enhancing interaction elements are added to the user interface design. For example, a button is added to the GUI that, when pressed by the user, will show the current status of the adaptive application or the adaption history, i.e. where am I and how did I get here?

Finally, in step six the overall application architecture as well as the data flows are specified in an abstract model by the software engineers. There is a choice of different modelling languages that can be used for this task. Both, architecture and data flows are reviewed by the experts from the socio-technical disciplines.

C. Software design, implementation, evaluation

So far, we have concentrated primarily on the requirements analysis and conceptual design phases because that is where the socio-technical concerns are explicitly visible. Later on, in the running software they will be represented in technical artefacts whose links to the socio-technical requirements may not be so obvious.

The development phases software design and implementation are carried out mainly by software engineers. The socio-technical domain experts are on stand-by for further questions and clarifications. The evaluation phase is a joint task of all involved disciplines. Software design, implementation, and the first part of the evaluation will be done in an iterative process consisting of consecutive cycles. When a new prototype of the application has been completed, the experts from the different disciplines in the development team will evaluate whether the prototype satisfies their requirements. This iterative process will continue until all requirements are satisfied by the prototype. Conflicting requirements may be detected and negotiated once more in this phase.

When a stable prototype has been achieved and agreed by the development team, a user evaluation will be conducted, which constitutes the second part of the evaluation phase. This requires experiments performed with real users that are not members of the interdisciplinary design team. The evaluation may involve simulated user scenarios, interviews, questionnaires, and usability studies with special devices such as eye-trackers and other laboratory equipment. Such an experimental evaluation may reveal weak points in the application design that require another iteration of the development process.

IV. RELATED WORK

There are only a few software development methods that explicitly focus on non-functional properties (though not in the sense of our socio-technical requirements) and cover the whole software development process. In the following we briefly present the three most relevant ones for our research.

In [3] an approach called FRIDA (From RequIrements to Design using Aspects) is proposed which aims at guiding the application developer through the phases of the software life cycle. FRIDA concentrates on both functional and nonfunctional requirements based on aspect-oriented modeling. Each non-functional requirement is represented by one or more aspects. Checklists are used to refine non-functional requirements at early-stages of the development life cycle and

to detect conflicting functional and non-functional requirements. The main difference to our approach is that FRIDA concentrates on the classic software-oriented non-functional requirements, while our focus is on orthogonal socio-technical considerations.

Reference [4] presents a coherent goal-driven development process as well as reusable quality characteristics that can be applied in software specifications. It is demonstrated that such quality patterns can be stored in a repository, from where they can be retrieved for reuse, tailored for different contexts and integrated with functional descriptions.

The authors of [5] propose an approach that integrates security concerns into systems engineering throughout the entire system development process. It builds on the Tropos methodology [6] that considers not only the system functional requirements but also non-functional requirements such as security, reliability, and performance. Tropos is based on the idea of building a model of the system that is incrementally refined and extended from a conceptual level to executable artefacts, by means of a sequence of transformational steps.

All of the three methodologies focus on non-functional requirements, but none of them assigns specific priority to the described user-centered socio-technical requirements. Certainly, our approach generally benefits from the published know-how of the three methodologies but cannot reuse them due to the very different nature of the addressed concerns.

V. Experiences and Conclusions

In order to evaluate our interdisciplinary development methodology for UC applications, we have performed three separate application development projects whereby each application is developed in two versions: The first version was developed without using the described methodology. Thereafter, the application is re-developed from scratch by applying the methodology. Each of the two versions is evaluated separately in extensive user experiments.

The three applications cover a rather diverse spectrum of UC scenarios: (a) a mobile self-adaptive social networking application that reacts to user context changes and can incorporate dynamically discovered services in the user environment, (b) an intelligent home application that supports elderly people who live alone in a private household, and (c) an RFID-based monitoring application that is able to track and record social contacts of persons, e.g. during a conference or in an office environment. Obviously, these scenarios raise plenty of challenging socio-technical concerns that need to be addressed in the development process.

The development of the second versions of the applications is work in progress. The prototypes are completed but still under evaluation. However, already we can see clearly that the new methodological approach involving experts from different disciplines, changes fundamentally the set of requirements as well as the functionality and the look-and-feel of the final application. Before developing the second versions with our new methodological approach, we did expect some substantial changes due to the involvement of the socio-technical disciplines. However, we did not expect that the assumption of an open, ubiquitous computing environment would introduce so many new non-technical requirements and concerns that computer scientists and software engineers alone would not have thought of.

A second immediate observation is that application development based on the new methodology takes roughly twice as long due to many review circles and resulting development iterations. Obviously, this observation has to be taken with caution because working with new methodology for the first time will always incur more overhead due to learning effort and discussions about the approach itself. A more substantial and detailed evaluation of the efficiency and effectiveness of our participatory design approach is ongoing work and will be the subject of a forthcoming report.

As part of our work general (unsolved) questions with software engineering methodologies have surfaced again: How does one measure the utility of a methodology? What are appropriate metrics? What kind of quality sensors do such metrics require? Probably only experience with many development projects over a large time span and a large spectrum of application scenarios will provide the answers to questions on how well a methodology works and how it compares to other approaches.

One conclusion is clear already: User-centered, context-aware, self-adaptive UC applications for sure require an interdisciplinary development methodology as an enabler for socially embedded and widely accepted solutions.

Acknowledgment

We thank all members of the VENUS project at the University of Kassel for their contributions.

References

[1] L. Chung and J. C. S. Prado Leite, On Non-Functional Requirements in Software Engineering, A.T. Borgida et al. (Eds.): Mylopoulos Festschrift, Springer LNCS 5600, pp. 363–379, 2009.

[2] ISO, Dialogue principles, ISO Standard 9241-110

[3] S. Bertagnolli and M. Lisboa. The FRIDA model. In Analysis of Aspect-Oriented Software (ECOOP 2003), July 2003.

[4] J. C. S. Prado Leite, Y. Yu, L. Liu, E. S.K. Yu, and J. Mylopoulos. Quality-Based Software Reuse. Proc. 17th International Conference on Advanced Information Systems Engineering CAiSE '05, Springer LNCS, vol. 3520, pp. 535–550, 2005.

[5] H. Mouratidis, P. Giorgini, and G. Manson. Integrating Security and Systems Engineering: Towards the Modelling of Secure Information Systems. Proc. 15th Int. Conf. on Advanced Information Systems Engineering, CAiSE '03, Springer LNCS, vol. 2681, pp. 63-78, 2003.

[6] J. Castro, M. Kolp, and J. Mylopoulos. A Requirements-Driven Development Methodology. Proc. of 13th Int. Conf. on Advanced Information Systems Engineering, CAiSE '01, pp. 108–123, 2001.

[7] D. Gefen, E. Karahanna, and D. W. Straub. Trust and TAM in Online Shopping: An Integrated Model. MIS Quarterly, 27 (1), 51-90, 2003.

[8] J. D. Lee and K. A. See, Trust in Automation: Designing for Appropriate Reliance. Human Factors, 46 (1), 50-80, 2004.

[9] V. Hammer, U. Podlesch, and A. Roßnagel. KORA – Eine Methode zur Konkretisierung rechtlicher Anforderungen zu technischen Gestaltungsvorschlägen für Informations- und Kommunikationssysteme. Infotech/I+ G, 21–24, 1993. (In German)

[10] J. Floch, C. Frà, R. Fricke, K. Geihs, J. Lorenzo, et. al.: Playing MUSIC - Building context-aware and self-adaptive mobile applications, Software Practice & Experience, John Wiley & Sons, http://dx.doi.org/10.1002/spe.2116, April 2012.