AFFINE for enforcing earlier consideration of NFRs and human factors when building socio-technical systems following agile methodologies

Mohamed Bourimi[1,2], Thomas Barth[2], Joerg M. Haake[1], Bernd Ueberschär[3], Dogan Kesdogan[3]

[1]FernUniversität in Hagen, Cooperative Systems, 58084 Hagen, Germany
[2]University of Siegen, Chair for IT Security, 57076 Siegen, Germany
[3]Leibniz Institute of Marine Sciences at the University of Kiel, 24105 Kiel, Germany

Abstract. Nowadays, various user-centered and participatory design methodologies with different degree of agility are followed when building sophisticated socio-technical systems. Even when applying these methods, non-functional requirements (NFRs) are often considered too late in the development process and tension that may arise between users' and developers' needs remains mostly neglected. Furthermore, there is a conceptual lack of guidance and support for efficiently fulfilling NFRs in terms of software architecture in general. This paper aims at introducing the AFFINE framework *simultaneously* addressing these needs with (1) conceptually considering NFRs early in the development process, (2) explicitly balancing end-users' with developers' needs, and (3) a reference architecture providing support for NFRs. Constitutive requirements for AFFINE were gathered based on experiences from various projects on designing and implementing groupware systems.

1 Introduction

Nowadays, a shift is taking place from single-user-centered usage to support multi-user needs and hence covering many collaboration measures and social aspects. The needed technical support for these users' activities in many important areas of our professional and leisure life activities is provided through collaborative applications also known as groupware as well as social software. According to Shneiderman and Plaisant *"an extrapolation of current trends leads to the suggestion that most computer-based tasks will become collaborative because just as most work environments have social aspects"* [1]. Thus, software systems and applications supporting collaboration are considered as socio-technical systems in the Computer-Supported Cooperative Work (CSCW) as well as Human-Computer Interaction (HCI) research fields [2]. Because socio-technical systems are characterized by complex scenarios which are mostly reflected e.g. in the user interface, HCI and CSCW also focus nowadays on human aspects of the development of computer technology in collaborative settings. While the goals of interaction are mostly covered by functional requirements (FRs), users' preferences (e.g. usability) and concerns (such as privacy

and security) are related to non-functional requirements (NFRs). According to [3], FRs define what the system does and therefore its functionality whereas NFRs define how a system has to be. Many CSCW and HCI key literature studied NFRs such as usability in socio-technical and the trade-offs, which could arise between them, e.g. privacy and awareness trade-offs in those systems. However, various literature state that current approaches do not adequately consider generally NFRs from the beginning in the development processes such stated in [3]. Thus, recently many development approaches especially in the area of socio-technical systems follow user-centered and participatory design in combination with agile methodologies in order to efficiently react on end-users' emerging needs and (change) requirements [4,5,6]. In our opinion, even when a given NFR is considered from the beginning (i.e. usability in user-centered or participatory design), it is mostly contemplated separately from other NFRs and factors. When considering that socio-technical systems mostly represent a special category of distributed systems that are known to be difficult to design and maintain, tensions could arise between project stakeholders (i.e. end-users and developers) especially in agile settings. Furthermore, current approaches often not explicitly address the gap of mapping NFRs into the underlying system architecture. In this paper, we present the AFFINE (Agile Framework For Integrating Non-functional requirements Engineering) *simultaneously* addressing these needs.
We first present identified needs in Section 2. Next, we describe our approach consisting of the AFFINE framework in Section 3 and our conclusions in Section 4.

2 Problem and Requirements Analysis

Software development processes can be seen as complex collaborative social processes. In order to reduce the potential complexity of these processes and assure the delivery as well as the quality of the products, many models (e.g. the well-known waterfall, prototyping, and spiral model) tried to structure the software development processes and define their behavior e.g. by introducing roles and defining software development life cycles. Latter include common phases like the requirements analysis, design, development, testing, and support phases. In contrast to the classical defined process models, agile process models and methodologies intend a better reaction on unexpected problems often by consideration of human factors. They are empirical processes that cannot be consistently repeated and therefore require constant monitoring and adaptation [7]. However, Balzert states in [8] that according to a coarse classification of the activities independently of a given development processes, one could generally differentiate between two main phases, namely, the *solution specification phase* and the *solution construction phase*. While most of the activities of the specification phase can be classified as requirements engineering activities, the activities of the construction phase target mapping a given solution specification to a concrete technical solution. Different software engineering practices recognized the critical importance of NFRs for the specification and construction of software systems in general. A classical work addressing NFRs is [3] state that software engineering practices concentrate on FRs, rather than NFRs. Furthermore, the authors cite that NFRs are generally stated informally during the requirements

analysis, are often contradictory, difficult to enforce, and to validate during software development process. Based on further literature, they state that not taking NFRs properly into account is acknowledged to be the most expensive and difficult to correct once the software has been implemented and thus, there is a need to deal comprehensively with such requirements during the system development process. The concrete needs we address in this paper were identified based on one of the long-running project CURE (Collaborative Universal Remote Education) we were able to follow. This project has a very representative character since its needs correspond to identified needs in other literature. The CURE platform was developed at the FernUniversität in Hagen (FuH) to support different collaborative learning and collaborative work scenarios [9]. The development process followed in CURE is an agile process called the Oregon Software Development Process (OSDP) described in [10]. Applying OSDP considered end-users' feedback of the participating departments at the FuH. Representatives of students and instructors from various disciplines such as mathematics, electrical engineering, computer sciences and psychology were participating in the usage and evaluation of the prototypes resulting from each OSDP-iteration. Even though OSDP considers conceptually NFRs in form of a NFR backlog, their consideration was not earlier enough to overcome drawbacks in the construction phase. In the case of CURE, responding to end-users wishes related to NFRs (e.g. usability of the web interface, performance of the synchronous communication means and awareness provision in the shared workspaces) was interrupted in order to meet the delivery, integration deadlines and budget. CURE was extended in various sub-projects (e.g. [11,12,13]). Most of these works were primarily concerned with improving NFRs which were classified as insufficiently covered by the developed system or tried to address new needs emerged through the usage of the system. Ambler states in [14] that NFRs and constrains are difficult to consider in projects following agile methodologies. A conceptual consideration of NFRs in the followed methodology avoids delegating their fulfillment to the intuition of involved people that could result in intentional or accidental negligence. Thus, we identify the need of ***conceptually enforcing the consideration of all relevant NFRs and possible trade-offs early in the development process (N1)***.

Involving end-users in an agile process could be very expensive. Especially when an agile methodology is followed in the end-users as well as developers are often experiencing continuous communication tensions. Developers are often asked to change, e.g., user interfaces or functionality, which seemed to be agreed upon earlier. Furthermore, on the one hand end-users and developers have different terminology for the same things or the same terminology for different things. On the other hand, members of the same development team might have different backgrounds and terminologies. This is also crucial in the case that different partners and/or distributed teams are cooperating in the same project. Communication problems are well known in the software engineering field and do not concern only agile methods. The same methodology may not be imposed to different stakeholder in the project, since involved parties may already have elaborated methodologies and processes as well as have different interests and goals (i.e. using their own software pieces or products etc.). Indeed, recent studies show that the most frequent failure source are communication problems with more than 70% [15] and that 33% of the projects are negatively affected or cancelled because of changing the requirements [16]. Based on

our experiences we argue that this is especially expensive when following an agile development process. Even though agility assure the close involvement of end-users, latter are mostly not experienced in communicating requirements to the developers [8]. Thus we identify the second need of ***explicitly balancing end-users' with developers' needs when following agile development method(ologie)s (N2)***.

The design and evaluation of socio-technical systems is still a challenge because of the exploratory nature of these applications [1]. Indeed, people involvement varies and the usage can range from occasional to frequent according to a given setting and circumstances. The same socio-technical system can lead to different evaluation results in different social environments [2]. The evaluation of socio-technical systems needs methodologies and approaches that allows for rapid and cost-effective development and usage of prototypes. Shneiderman and Plaisant mention that *"while software engineering methodologies are effective in facilitating the software development process, they have not always provided processes for studying the users, understanding their needs, and creating a usable interface"* [1]. Depending on the project specific situation, development costs need often to be reduced. Software should not be built from scratch each time in the development process. In [17], Grudin addresses challenges for groupware developers and suggests that adding new functionality to an accepted application is more adequate than developing a new application. This is a typical case when building many socio-technical systems or while their evolution. At a first glance, adding new functionality and enhanced interaction possibilities to existing systems seems attractive. However, adding new functionality often requires adding and modifying a lot of source code. This often complicates the API and requires a redesign of the domain model. Furthermore, Paech et al. argues in their position paper [18] that FRs and NFRs as well as architecture should not be separated. The emerging changes are especially crucial when considering costs in terms of (re-)design, implementation and retrofitting costs. Thus, added functionality to socio-technical systems is reflected in growing complexity of their classes and/or components. Thus, the extension or retrofitting and the integration of new components in these systems represent realistic scenarios, which have to be considered in terms of development costs. However, it is important that by freezing changes, the design of the system stays extendable for future extensions and retrofitting. Thus, we formulate the third need as follows: ***The development method must be supported at the architectural and construction level to assure meeting N1 and N2 at minimal cost. A Kind of reference architecture providing support for NFRs is needed (N3)***. While **N1** is more concerned with the specification phase of a given socio-technical system and **N3** with its construction phase, **N2** still overlapping both phases when following an agile methodology. S*imultaneous* consideration of **N1-N3** is therefore required.

3 Our Approach: AFFINE

Introducing an agile method at the level of the development process is the key to satisfy **N1** and **N2** at the organizational level in our opinion. In order to reduce the complexity of the involvement of our method in various phases of the followed

development process in a given project, we propose as an integral component Scrum [28]. Scrum can be seen as a process for empirical control of software development, which helps in handling changing requirements more efficiently by considering human factors in the development process of both; customers (in our case end-users) and project stakeholders in general. Based on our experiences in various projects, the main strengthens of Scrum consists of (1) the simplicity in terms of roles defined (Scrum master, product owner, and development team), development steps to be followed (e.g. development periods called sprints), documentation to be produced (e.g. sprint backlog), and meetings to be held (e.g. daily Scrum), (2) balancing the needs of the customers and developers through consensus enforcement for a given deliverable and continuous communication (e.g. in the daily Scrum meetings), (3) creating awareness on ongoing project tasks (also in daily Scrum e.g.), and (4) allowing for better as well as faster handling of detected, non-expected problems during the development process, which generally results (by right application) in better acceptance of the delivered product with low costs.

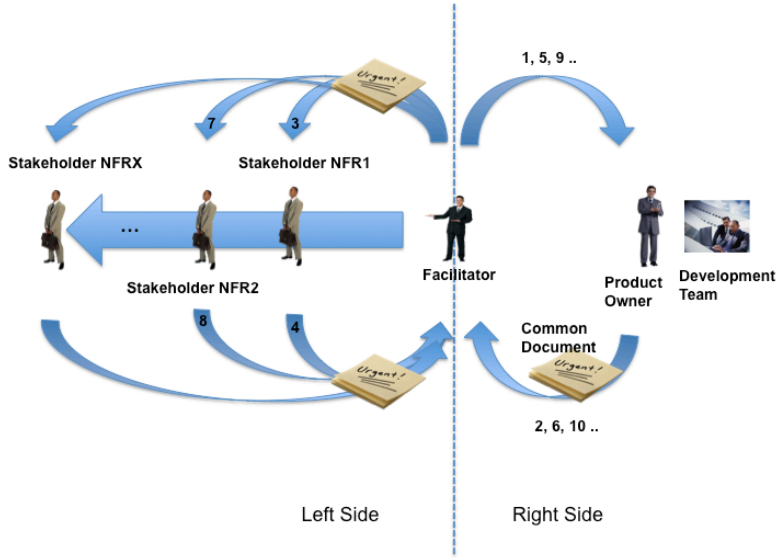

Figure 3.1: The Scrum based AFFINE method.

The right side of Figure 3.1 represents broadly a typical Scrum development procedure. A facilitator (in our case the Scrum master) as well as the product owner and development team interact with each other in order to drive the product development. This interaction is represented as loop involving them all together. Since the facilitator moderates the interaction, we represent such loop as an arrow starting and ending in the facilitator role. In Scrum, a sprint backlog document might be updated in such loop. Numerating this loop in our representation does not imply that the interaction is carried out in a giving sequence inside it. The left side of the same Figure shows our extension of a typical Scrum process to enforce the earlier consideration of NFRs. There, we introduce the role of a NFR stakeholder (mostly experts), who is concerned with the fulfillment and consideration of a respective

NFR. The same facilitator has to moderate the circulation of the common document to the first stakeholder (according to prior prioritization), who has to update the document (e.g. adding warnings, requirements or changing them etc.). The structure of the common document and its content have to be defined from the project involved parties. However, circulating only one document, which contains all needed information for the development of given product, should avoid potential inconsistencies and information lost. After updating the document (at least by annotating that it was reviewed and maybe admitted without changes from the respective stakeholder), the document returns to the facilitator(s) and loops again over the left side. By admission without changes, the facilitator might shorten this iteration and directly forward the document to the next stakeholder. If any change happens in late circulations at the level of a stakeholder, the document returns to the facilitator, has to be circulated in the right side, and finally has to begin the circulation at the first stakeholder at the left side. We suggest the following informal steps for N1 and N2:

1. Involvement of all stakeholders of the project and introducing the role of the facilitator (one or more according to the project setting).
2. Goals or use cases (UC) identification of the intended processes (by defining the set of FRs). The facilitator has e.g. to guarantee the same terminology is used and has to detect miss-satisfaction signs in the different phases.
3. Alignment of all NFRs that have to be considered prioritizing them according to the project goals or UCs.
4. Responsible and experts for each goal or UC as well as NFR have to be chosen.
5. Circulating a single document (to avoid syncing various documents) containing the set of goals or UCs and their specification and modeling (by considering aligned FRs and NFRs at the same time). For this, UML or similar notations could be helpful in order to estimate efforts. The circulation has to be performed by the facilitator according to the priority of the NFRs. If a breakdown is identified in the circulation loop, the document has to be send back to the responsible of the first affected NFR (ordinarily with higher priority). If many NFRs are simultaneously affected, a meeting of the responsible and experts has to be organized. When conflicts arise, the facilitator intervenes in order to reach consensus. Since the facilitator is normally only a supporting role, his main goal consists in delivering the result while preserving satisfaction of end-users and project stakeholders. However, the final decision has to be made by the responsible(s) or at least by the coordination entity of the project.
6. The circulation ends when reaching the goals i.e. by implementing the UCs and testing them (also through the end-users).

Those steps have to be executed for each project iteration. If the project is organized according big work packages, following a divide and conquer methodology could be helpful. In order to optimize the requirements gathering, user-centered design and modeling steps (for instance by using established methods like prototyping) and UML or ER diagrams (as mentioned before) could be useful.

Finally, we want to mention, that the facilitator does not represents a critical point in this procedure. Any person familiarized with development activities should be able to act within this process as a facilitator. Further, if Scrum is integrated as an agile method, the Scrum certification exam ensures needed qualification of a facilitator. Related to **N3**, SOA is currently assessed as the next step forward in the design,

development, operation, and organization of large-scale distributed systems (see e.g. [22]). Characteristics like loose coupling, discovery of artifacts during design/run time, and the ability to reuse services to enable efficient adaptation of a system to changing requirements (e.g. changes in users behavior, processes) are not supposed to be provided by an architectural approach for the first time. Learning from preceding approaches, characteristics like the commitment to open standards and the separation of architectural concepts from their technical implementation led to a widespread acceptance of service-oriented principles in commercial as well as scientific communities. Since in our context NFRs are in the focus of attention, the inherent possibility of tailoring an SOA at design time according to the actual needs by carefully selecting the specifications to implement is an adequate means to realize these requirements [22]. Beyond this, NFRs like usability and performance in socio-technical systems can only be assessed during runtime and in close cooperation with the end-user. As already discussed in this paper, an early consideration of NFRs during the lifecycle of an SOA leads to reduced development cost. Integrating different stakeholders into the development of an SOA is one approach to handle this; a Service Life Cycle Model focusing on SOAs stakeholders as a prerequisite for governing an SOA throughout its lifecycle is presented in [23]. Since NFRs are crosscutting concerns, the positive synergy between aspect-oriented programming (AOP) techniques and SOA for satisfying NFRs implementation was beneficial in our case. The architecture of the CURE-based sub-projects described in the related publications supports different kind of clients (i.e. Eclipse RPC thick client for the collaborative design editor [11], normal and AJAX browser clients for the retrofitted CURE [12], and mobile as well as ubiquitous clients for the ubiquitous CURE [13]) with the same SOA/AOP layer. Thereby, different kinds of architecture families also are supported (based the on client-server model, replicating or P2P). Surely, this is due because the context of these sub-projects could be satisfied with such single layer. Nevertheless, if different contexts have to be supported, various instances of the SOA/AOP layer could be deployed. So we mean that our AOP/SOA-based generic architecture meets **N3** with a high genericity.

4 Conclusion

In this paper, we proposed the AFFINE framework that aims at *simultaneously* addressing three needs when developing socio-technical systems by following agile methodologies. The three needs were identified from the long-running project CURE as well as based on relevant HCI and CSCW literature gathered experiences. The main idea of AFFINE is to use an agile method including Scrum as an integral part. The agile method is user-centered and considers human factors, which could affect the success and acceptance of the developed socio-technical systems. The method enforces conceptually the earlier consideration of NFRs while the suggested supporting architecture provides a generic reference architecture for developing socio-technical systems in agile development settings. Furthermore, AFFINE is independent from a specific software development process and applicable for different phases of the followed process in a given project. The concrete suggestion to use SOA and AOP

at the technological level showed their advantages in first evaluations. The proposed framework is now successfully being applied in the ongoing work of many projects led by us as first empirical evaluations show. Future work aims at collecting more experiences and refining AFFINE.

5 References

1. Shneiderman, B., Plaisant, C.: Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition). Pearson Addison Wesley (2005).
2. Gross, T., Koch, M.: Computer-Supported Cooperative Workspace. Oldenburg (2007).
3. Chung, L., Nixon, B.A.: Dealing with non-functional requirements: three experimental studies of a process-oriented approach. In: ICSE '95, ACM (1995).
4. Jokela, T.: Assessment of user-centred design processes - lessons learnt and conclusions. In: PROFES. (2002).
5. Schümmer, T., Lukosch, S., Slagter, R.: Empowering end-users: A pattern-centered groupware development process. CRIWG 2005. Vol. 3706, Springer (2005).
6. Lieberman, H., Paterno, F., Wulf, V., eds.: End User Development. Springer (2006)
7. Highsmith, J.: Agile software development ecosystems. Addison-Wesley (2002).
8. Balzert, H.: Lehrbuch der Softwaretechnik, Basiskonzepte und Requirements Engineering, 3ed., Spektrum (2008).
9. Haake, J.M., Schümmer, T., Haake, A., Bourimi, M., Landgraf, B.: Supporting flexible collaborative distance learning in the cure platform. Vol. 1, IEEE Computer Society (2004).
10. Schümmer, T., Slagter, R.: The oregon software development process. (2004).
11. Bourimi, M.:Collaborative design and tailoring of Web based learning environments in CURE. In CRIWG 2006. LNCS 4154, Springer (2006).
12. Bourimi, M., Lukosch, S., Kühnel, F.: Leveraging visual tailoring and synchronous awareness in Web-based collaborative systems. In CRIWG. Volume 4715, Springer (2007).
13. Bourimi, M., Kühnel, F., Haake, J., Abou-Tair, D., Kesdogan, D.: Tailoring collaboration according privacy needs in real-identity collaborative systems. In: CRIWG. (2009).
14. Ambler, S.W.: Beyond functional requirements on agile projects. Dr. Dobb's Journal 33(10) (2008).
15. Huth, S.: Marktstudie probleme und fehler im requirements-engineering, www.sigs-datacom.de/wissen/artikel-fachzeitschriftel/artikelansicht.html ?tx_mwjournals_pi1%5BshowUid%5D=2479 (Accessed 2010).
16. Emam, K.E., Koru, A.G.: A replicated survey of it software project failures. IEEE Softw. 25(5) (2008).
17. Grudin, J.: Groupware and social dynamics: eight challenges for developers. Communications of the ACM 37(1) (1994).
18. Paech, B., Dutoit, A.H., Kerkow, D., Knethen, A.V.: Functional requirements, non-functional requirements, and architecture should not be separated. Technical report, Proceedings of the International Workshop on Requirements Engineering (2002).
19. Schwaber, K.: Scrum overview. http://codebetter.com/blogs/darrell.norton/pages/50339.aspx (Accessed August 2010).
20. Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall PTR (2006).
21. Gu, Q., Lago, P.: A stakeholder-driven service life cycle model for SOA. In: IW- SOSWE '07: 2nd international workshop on Service oriented software engineering, ACM (2007).