VENUS meets SEMAT– How do they compare?

Kurt Geihs, Christoph Evers, and Stefan Niemczyk

EECS Department
University of Kassel
Wilhelmshoeher Allee 73
D-34121 Kassel, Germany
[geihs|evers|niemczyk]@uni-kassel.de

Abstract SEMAT (Software Engineering Methods And Theory) is an initiative to define a generic foundation for software engineering as a rigorous discipline. The so-called SEMAT kernel provides a thinking framework for software engineers that is not constrained to certain methods and processes but aims to encompass all kinds of proven principles and best practices. Our own interdisciplinary VENUS development method is designed to achieve similar generality and compatibility objectives, although the chosen application domain in VENUS has a much narrower scope. In this paper we compare the VENUS development method with SEMAT. The main contributions are positioning the VENUS development concepts within the SEMAT conceptual framework, and investigating whether SEMAT is an appropriate framework for dealing with inherently interdisciplinary development processes. In the end we present suggestions for the improvement of both approaches.

Keywords: Software Engineering, SEMAT Kernel, Software Development Method, Socially Aware Computing

1 Introduction

Ubiquitous Computing (UC) interweaves information and communications technologies with our daily living environment. It is a salient characteristic of UC applications that such applications collect, store, process, and communicate personal information about the user's context – often transparently and imperceptibly for the user – in order to realize an adaptive application behavior. This can lead to conflicts regarding the social embedding of the technology. Some examples: The inherent lack of transparency in automated activities can lead to a reduced level of trust with the user; liability issues in self-adaptive activities may be problematic from a legal perspective; usability concerns in UC applications require new user interface designs that do not overstrain the user. It has been one of the fundamental claims of project VENUS that self-adaptive UC applications will only have real impact and find widespread acceptance if these socio-technical concerns are taken seriously and appropriate interdisciplinary design methodologies are available that support the development of socially aware software systems. This requirement is also expressed in other recent publications, such as [16, 14].

The VENUS project has developed a discipline-overarching method for the design and evaluation of socially aware self-adaptive UC applications. One of the design goals was to maintain compatibility with existing software and discipline-specific design methods. In the past there have been several attempts to specifically represent socio-technical aspects of business processes and organizational structures in software development methodologies [2]. Moreover, frameworks were presented that particularly focus on the systematic treatment of non-functional requirements in software engineering processes (e.g. [4, 15, 19]). However, to the best of our knowledge, none of those approaches takes into account the specific requirements of context-aware self-adaptive UC applications that collect and process vast amounts of sensitive user data.

SEMAT (Software Engineering Methods And Theory) is an initiative to define a generic foundation for software engineering as a rigorous discipline [12]. The declared objective for the so-called SEMAT kernel is to provide a thinking framework for software engineers that is not constrained to certain methods and processes but aims to encompass all kinds of proven principles and best practices. The SEMAT initiative was founded in 2009 by Jacobson, Meyer, and Soley, all three internationally renowned software engineers. Very rapidly the initiative has received widespread attention and support within the software engineering community. A strong indicator for this trend is the large number of signed-up supporters and the various workshops on SEMAT issues at major conferences.[1] Over the years SEMAT has produced, among many other documents, a comprehensive specification as a submission in response to the OMG Request for Proposal on a "Foundation for the Agile Creation and Enactment of Software Engineering Methods" [8].

"How do VENUS and SEMAT compare?" is an obvious and very relevant question to ask here. In order to shed light on both new methodological endeavors, in this paper we will discuss whether and how the methodological approach of VENUS with its explicit focus on socially aware software development can be placed in the conceptual frame of SEMAT which aims at genericity and compatibility. We would like to explore whether the intrinsically interdisciplinary approach of VENUS can be mapped to the currently available SEMAT concepts, and whether the SEMAT framework provides inspirations to improve the VENUS approach and its presentation to the software engineering community. Hence, the main contributions of this paper are (1) positioning the VENUS development concepts within the SEMAT conceptual framework in order to facilitate the comprehension of the VENUS approach, to foster its take-up, and to derive improvement suggestions as feedback to the VENUS researchers, and (2) investigating whether SEMAT is an appropriate framework for dealing with inherently interdisciplinary development processes in order to provide feedback to the SEMAT developers.

The remainder of this paper is structured as follows: Section 2 presents an overview of the SEMAT specifications. Section 3 briefly reviews the principal constitu-

[1] See www.semat.org for details.

ents of the VENUS development method, as far as they are needed for the following discussions. In Section 4 we will position and compare VENUS in respect to SEMAT. Section 5 summarizes our conclusions and formulates open questions for future research.

2 SEMAT

In the following we present a short overview of the main concepts of SEMAT. It is based on the three documents [8, 12, 13]. Text quotations taken from these documents are written in italics. Here we focus on a brief summary of the key technical contents of the SEMAT initiative. More details can be found in the cited literature.

One of the key motivations for the SEMAT activity is the observation that today there is a large number of software engineering methods without a clear understanding how they can be compared and combined and what the consequences of preferring one over the other are. There is a lack of a widely accepted common ground for software engineering practices, i.e. a conceptual basis that not only satisfies the interests of theoreticians but also provides a guideline for improving the performance of real-life software development projects.

SEMAT has two – largely independent – main objectives: (i) to come up with an extensible and practical kernel of essential elements that are applicable to all software development efforts, and (ii) to define a solid theoretical foundation. While work on the more pragmatic first objective has made significant progress, work on the second objective is in a much less mature state: No common theoretical foundation has been agreed upon so far.

The SEMAT kernel is meant to be agnostic to any particular software engineering method and practice. In particular, among other contributions it provides a *common ground for the discussion, improvement, comparison, and sharing of software engineering methods and practices* as well as *a framework for teams to assemble and continuously improve their way of working by the composition of separately defined, and sourced, practices* [12]. These aspects of the SEMAT kernel are of particular interest from the viewpoint of the VENUS developments and will be further investigated in Section 4.

The kernel is organized into three areas of concern, i.e. Customer, Solution, and Endeavor. The *customer concerns area contains everything to do with the actual use and exploitation of the software system to be produced. The solution area of concern contains everything to do with the specification and development of the software system. The endeavor area of concerns contains everything to do with the team, and the way they approach their work* [8]. Fig. 1 illustrates the three areas of concern and sets the context for the following two figures.

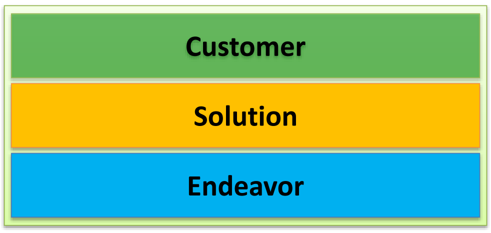

Fig. 1. SEMAT areas of concern

Within these areas of concerns the kernel distinguishes *things to work with*, called *alphas*, and *things to do*, called *activity spaces* [8]. An *alpha* is defined as *an essential element of the software engineering endeavor that is relevant to an assessment of the progress and health of the endeavor. Alpha is an acronym for 'Abstract-Level Progress Health Attribute'* [8]. The *alphas* represent *things that a team will manage, produce, and use in the process of developing, maintaining and supporting good software* [8]. Fig. 2 and Fig. 3 illustrate the two groups of "things". Fig. 2 shows the *things to do* for all software development projects. Fig. 3 shows the *things to work with* during software development (i.e. *alphas*) and how they relate to each other. The differently colored boxes represent the three areas of concern, i.e. Customer, Solution, and Endeavor (from top to bottom), as shown in Fig. 1. Kernel elements have specific states *that represent their progress and health* in the course of a development project. It is important to understand that the kernel views *software development not as a linear process but as a network of collaborating elements* [12]. This latter statement appears to be consistent with the VENUS methodology approach. We will come back to this issue in Section 4.

As one of the main objectives of SEMAT is the practicality of its concepts, the term "practice" and "method" play an important role in the framework [8]: *A practice provides a systematic and verifiable way of addressing a particular aspect of the work at hand.* Further, *practices are presented as distinct, separate, modular units, which a [development] team can choose to use or not to use.* Practices are described using the kernel elements. A method is defined as *a composition of practices forming a [...] description of how an endeavor is performed.* The authors of SEMAT emphasize that there are many practices around and that a development team is free to choose and compose the practices that best match its project [12]: *The kernel allows you to add practices [...] to build the methods you need.*

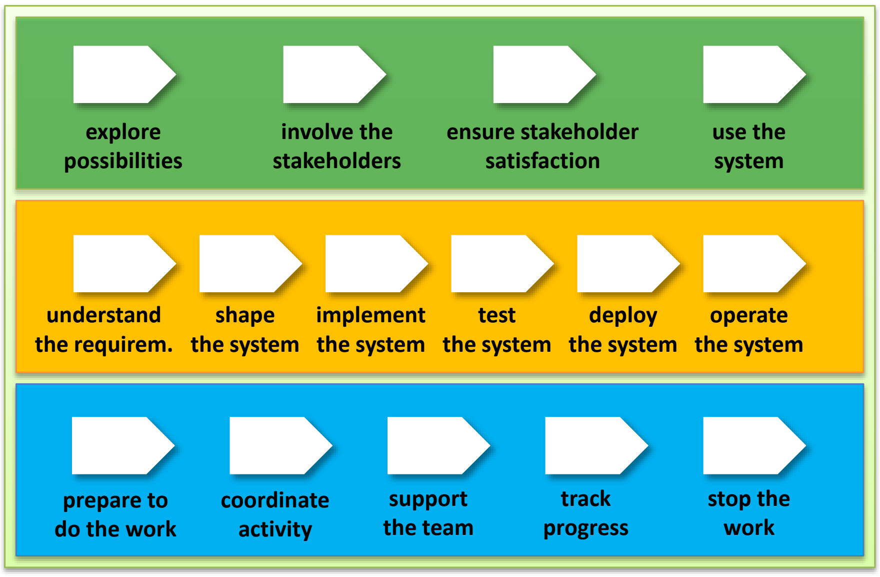

Fig. 2. SEMAT Things to do [12]

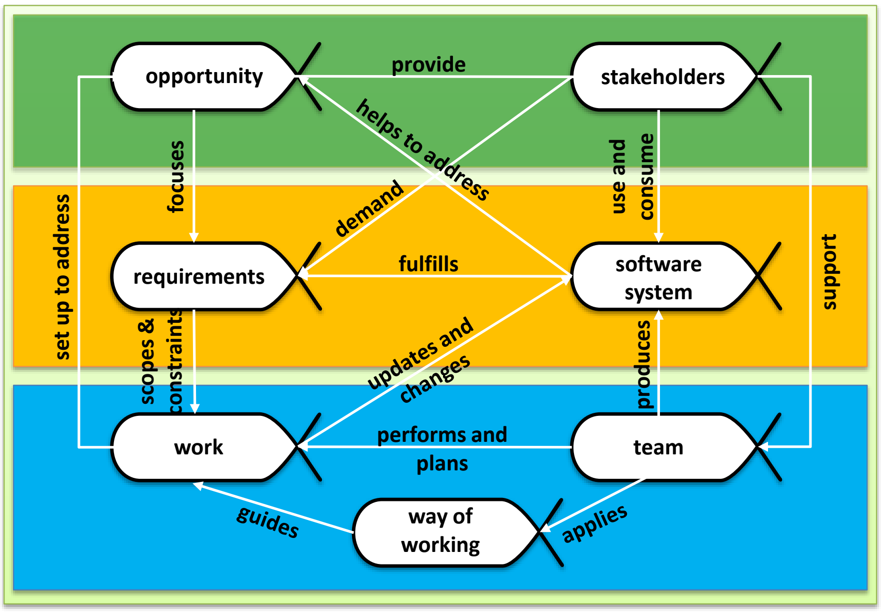

Fig. 3. SEMAT Things to work with (called Alphas) [12]

3 VENUS

For a detailed presentation of the VENUS development method the reader is referred to [7, 10]. Here we summarize only the key concepts, as far as they are needed to understand the following discussion, i.e. in order to make this paper self-contained.

The VENUS development method is based on an iterative development model that covers the well-known software engineering lifecycle, i.e. analysis, design, and evaluation. More precisely, analysis is divided into demand analysis and requirements management, while design comprises conceptual design as well as software design and implementation. In the final evaluation phase we test and evaluate the produced software by means of user studies and simulation experiments. In addition, interdisciplinary evaluation of intermediary results takes place in all phases of the development process. This can trigger additional iterations in the development phases.

Fig. 4 illustrates the development phases and the iterative character of the methodology. It emphasizes that interdisciplinary development in project VENUS has focused on social awareness in respect to legal compatibility, usability, and trust.

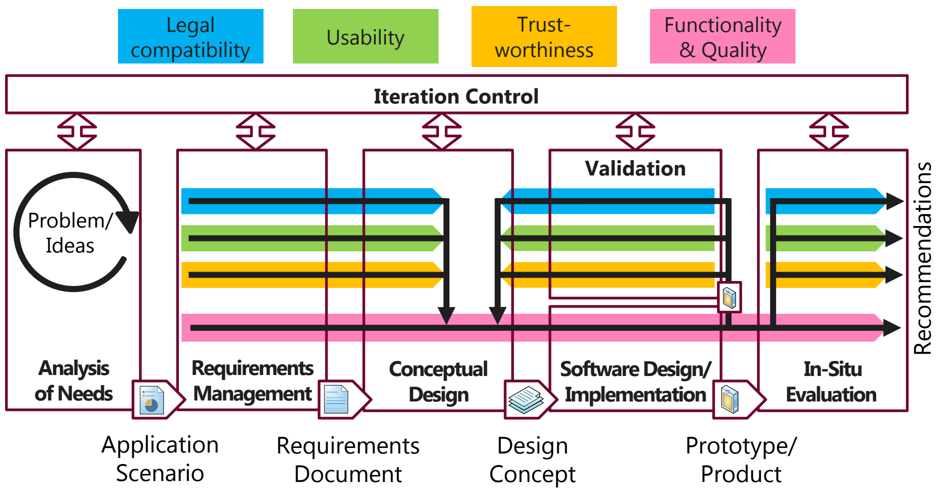

Fig. 4. Overview of the VENUS development approach [9]

Analysis of Needs: The first activity in the development of a UC application is the profound analysis of the problem by developing specific application scenarios. These provide a common understanding about the required application functionality and usage context. Application objectives are derived in workshops that involve experts and users and narrowed down to application scenarios and usage requirements. Furthermore, this activity may also involve the derivation of requirements related to appropriate business models and legal constraints.

The derived scenarios are validated by potential users of the applications. The objective of this validation activity is to discover as early as possible misunderstandings about the functionality, usage, and socio-technical embedding of the application under development.

Requirements management: This activity needs to solve two important problems raised by the interdisciplinary design of application software: First, we need to derive coherent requirements from the different sets of heterogeneous disciplinary requirements. Second, abstract socio-technical normative requirements and constraints need to be transformed into concrete technical requirements that are taken up in the design of the software.

Coherent requirements across disciplines can only be achieved if there is a common understanding of discipline-specific languages and terminology. This is a prerequisite for discussing the harmonization of requirements and resolution of conflicting requirements, e.g. by defining priorities or agreeing on compromises.

In order to derive concrete technical requirements from normative legal or ethical concepts we apply a stepwise refinement process which is based on a method called KORA [11, 20]. KORA was developed initially to translate legal requirements into technical ones when designing technology. In project VENUS this method was adapted and extended to include the transformation of abstract normative concepts from the fields of usability and trust management. At the end of the requirements management activities one has a set of coherent socio-technical design requirements expressed in the language of software engineers [10].

Conceptual design: The derived requirements from the previous step are the starting point to produce a conceptual design of the application. This is done in five steps that may be executed iteratively whereby each step can lead to feedback and a return to earlier development activities.

In step 1 use cases will be specified [18]. These serve as input for the development team to check that all requirements are taken into account correctly. Step 2 involves the identification of functional elements and data structures of the application. The next step is to show how these elements will be reflected in the specified use cases. Flowcharts and sitemaps [5] may help to visualize the sequence of user interactions and the relevant functions and data items. All discipline experts contribute to the validation of the results of this activity. The fourth step comprises the development of a preliminary graphical user interface design which is used to check whether and how socio-technical requirements have found their way into the interactions between application and user. Finally, step 5 includes the specification of the software components of the application, their interfaces and data structures, e.g. using a language such as the Unified Modeling Language (UML) [17].

Software design / implementation: The resulting specification, which reflects all socio-technical requirements, is passed on to the software engineers who will take it as a starting point for the software design and implementation phase. Besides conventional software engineering techniques model-driven as well as agile approaches can be applied here. A model-driven approach has the advantage that intermediary models capture already the more or less complete application functionality and properties. Thus, these intermediary results may be fed into tools that may test the abstract design for certain desired properties, such as correctness conditions or performance properties. However, the model-driven approach requires sound skills in suitable modeling languages which may be an obstacle if you are not a software engineer. An agile approach has the advantage that partial prototypes are built that can be studied and discussed by users without requiring a deep knowledge of software modeling techniques. However, the effects of concepts that overarch the whole application might not be visible in early, partial prototypes.

In-Situ Evaluation: Evaluations are performed in all phases of the software development process to test the functional correctness of the software and the social awareness criteria. This includes evaluation activities aiming at the user scenarios after requirement analysis and user interface prototypes after conceptual design, as well as functional and integration tests after software design and implementation.

In the final phase of the proposed software development process the produced implementation is evaluated in situated use by real users [3, 5]. The objective is to check in a realistic usage environment whether the developed application satisfies the stated expectations and requirements. Simulation studies may be performed, e.g. in a laboratory, in order to simulate real life situations and evaluate aspects such as usability, performance, and robustness.

We claim that although project VENUS specifically targets UC application scenarios, the developed methodology – aiming at a systematic social embedding of technology in order to assure user acceptance – can be applied to other application scenarios as well where social awareness plays an important role. Admittedly, this claim needs further confirmation by other project work.

4 VENUS and SEMAT

In this section we apply the SEMAT conceptual framework to the VENUS development method. Our objective is to highlight the particularities of VENUS and by doing so to evaluate the generic approach of SEMAT.

As a running example we use an application called Meet-U [6, 7] that is one of three joint demonstrators in the VENUS project. Meet-U was developed in two versions, i.e. the first one without and the second one with the VENUS development method. Thus, version 2 was influenced explicitly by social awareness concerns.

Meet-U is a mobile application that maintains a social network for a group of users. It supports the user in planning and performing joint recreational activities with friends. Based on user preferences appropriate activities are suggested, dates are negotiated and coordinated, navigation to events is provided, and useful services in the environment of the user are dynamically discovered and bound to the application. All of this happens in the application in a self-adaptive and context-aware manner depending on e.g. user location, device status, and availability of friends. Fig. 5 gives an impression of the user interface of Meet-U.

The adaptation manager in Meet-U uses context information obtained via the built-in smartphone sensors and other information sources (e.g. contacts and calendar) in order to reason about appropriate application adaptations and service bindings. This happens automatically and transparently without user interaction or notification. Thus, dynamic ad-hoc service access and the transfer of user data to services may imply a legal problem if services misuse personal data. To avoid such unintended leakage of personal data, i.e. to support the general legal obligation of information self-determination, users should have means to find out about and control the information that the adaptive application exchanges with a service. This requirement and many other similar sociotechnical requirements were not available in Meet-U Version 1, but designed into Version 2 using the VENUS development method.

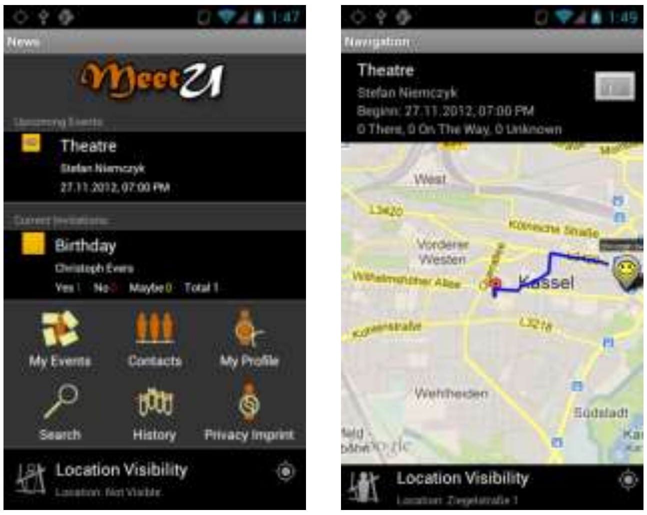

Fig. 5. Two screenshots of Meet-U

In the following discussions the term "VENUS" – unless explicitly denoted otherwise – refers to the interdisciplinary software development method of VENUS. Furthermore, we use *italics* when we refer to the SEMAT terminology.

4.1 Areas of Concern

Looking at Fig. 1 which displays the SEMAT areas of concern, VENUS addresses these areas too, but with different intensity. Moreover, it seems that "social awareness" or "social embedding" might as well represent a separate area of concern that has a cross-cutting nature and does not have suitable representations in the SEMAT "Things to work with" and "Things to do". Let us explore this thought by using the development of Meet-U as a test case.

Clearly, Meet-U is concerned with a *solution* that tries to grasp a specific *opportunity*, i.e. a *software system* for social networking on mobile devices that is built according to a large set of functional and non-functional requirements. As Meet-U is a research demonstrator, the *stakeholders* on the *customer* side are less influential during the development, i.e. their major role is evaluation of the intermediate software designs and the completed *software system*. A *team* of disciplinary experts derives the *requirements* for the Meet-U design that is transformed into a conceptual model and then into an implementation by the software engineers in the *team*. The VENUS methodological approach combines different disciplinary practices into a multi-disciplinary *way of working*.

As can be seen from these explanations, the SEMAT *alphas* appear rather naturally when describing the VENUS approach to developing Meet-U. The state of these *alphas* are implicitly advanced, monitored, and evaluated during the VENUS development activities. In a commercial setting (and even in a research project with given deadlines), certainly it would have been beneficial to monitor and control the progress of the alphas more systematically, as described by SEMAT. Hence, we propose that the specification of the VENUS approach should refer explicitly to the appropriate *alphas*.

Let us look at the *customer area of concern*. Since Meet-U is a research-oriented software prototype that specifically aims at experimenting with support for social awareness, the focus of the *work* (on Meet-U) is not so much on identifying *opportunities* and satisfying *stakeholders'* expectations, i.e. in Meet-U there is so far no emphasis on business-oriented customer concerns. Naturally this would change if we would aim at developing Meet-U into a commercial product.

As we understand the SEMAT kernel, there is one shortcoming with the *Things to work with* in the *solution area of concern*: Modern software engineering always tries to (re)use existing software components, modules, library functions, data structures, design patterns etc. For example, in Meet-U we rely on existing trust-enhancement components and we have identified candidates for interdisciplinary design patterns that show how to address and satisfy specific legality requirements in the software development process, e.g. the basic right for informational self-determination. (Chapter 16 of [7] discusses selected interdisciplinary design patterns for UC.) From our point of view it would be advisable to include in the SEMAT kernel one or more *alphas* appropriately representing "the identification and integration of existing solutions", e.g. legacy applications, standard components, design patterns, and more.

SEMAT has three areas of concern for software development, i.e. *customer*, *solution*, and *endeavor*. In VENUS we emphasize the importance of the social embedding of UC solutions: Not only the customers themselves pose requirements and demand certain features, but also the societal environment typically postulates and adheres to abstract ethical or legal norms that – at the end of the day – translate into concrete technical requirements for the software solution. (Remember the remarks made above on information self-determination.) In the proposed SEMAT kernel there is no area of representation for these concerns except for the *customer area of concern*. This is insufficient to model the origin of social awareness requirements that may not at all be conforming to the customer's demands. We claim that "social embedding" is an area of concern on its own that cannot be neglected – at least for certain types of application domains.

What would be the *things to do* and the *alphas* in this new area of concern? Concerning the *things to do*, we need to look at how VENUS manages the sociotechnical concerns during the development process (cf. Section 3). Thus, the abstract norms and constraints relevant for the application have to be identified, stepwise translated into concrete technical requirements, and evaluated throughout the whole development activities. To find applicable interdisciplinary patterns and reusable solutions would also belong to the *things to do* in the *area of concern* called social embedding.

Concerning the required additional *alphas*, at least an *alpha* would be needed representing the “adoption of sociotechnical norms and standards” that determines the social awareness context for the application. Another *alpha* in this area should represent the “application of non-technical benchmarks, conformance tests, evaluation criteria, and the like” that are valid and binding independent of any concrete application scenario. For example, such an *alpha* should be used to monitor and control the degree of conformance to specified usability standards or general legal obligations, as they were addressed in version 2 of the Meet-U application.

4.2 Practices in VENUS

SEMAT defines a method as a *composition of practices*, and *a practice provides a systematic and verifiable way of addressing a particular aspect of the work at hand* [8]. This is very much in line with the viewpoint of the VENUS development method. VENUS does not prescribe the use of specific practices in order to design, implement, and evaluate socially aware applications. Actually, VENUS is an example for the flexible composition of different practices.

Experiences with the development of three different VENUS demonstrators (called Meet-U, Connect-U, and Support-U [7]) reflect this flexibility. For example, while KORA (cf. Section 3) was used in all demonstrators to elicit concrete technical requirements from abstract normative regulations, different practices were used for the software design and implementation activities, e.g. agile techniques and more conventional development approaches. Clearly, the developers of the UC demonstrators might as well have used the VENUS development method in combination with model-driven design and implementation techniques within the frame of VENUS.

Obviously, the three social awareness dimensions of VENUS, i.e. usability, legal compatibility, and trust, do not at all cover the whole spectrum of social awareness concerns. The inclusion of other disciplines, such as sociology or ecology, would certainly bring in additional, domain-specific practices into the software development process. Most of them would primarily be applied during the requirement analysis and management, but could also have direct influence on the conceptual design of the application.

Thus, SEMAT provides a clarifying guideline by viewing practices as *distinct, separate, modular units that a team can choose to use or not to use* [8]. The choice of practices defines the *way of working* of the development *team*. It would be a very helpful exercise for VENUS to collect and describe all practices that have been considered in the multi-disciplinary software development efforts. This has not been done in a stringent way so far. It is to be expected that building such a repository of practices would yield insights in frequently used elements and thus lead to the definition of interdisciplinary and reusable abstract solution approaches for typical problems in the design of socially aware application software, ideally supported by a set of reusable general interdisciplinary design patterns. Such a set of interdisciplinary design patterns for UC applications is described in a forthcoming technical report [1].

5 Conclusions

VENUS has aimed at methodological support for the development of socially aware ubiquitous computing applications. It is built on the collaboration of disciplinary experts and leads to a systematic integration of non-functional requirements into the software development process. VENUS is agnostic to particular styles and methods used during software development.

The SEMAT initiative has taken a fresh look at software engineering methods. It has defined a kernel of elements that help to structure the way of working of software development teams in order to ultimately improve the way software is developed. While SEMAT is a generic and extensible framework for all kinds of application domains, VENUS focuses on context-aware, adaptive ubiquitous computing applications on mobile devices and their embedding into the social environment.

As we have discussed in this paper, the SEMAT philosophy of being agnostic to any concrete software engineering practices and methods matches well with the VENUS approach. We have argued that VENUS should think about building a collection of practices that can be used for the development of socially aware applications. This collection should be underpinned by a set of re-usable interdisciplinary design patterns. Furthermore, SEMAT's emphasis on defining states for the different alphas and monitoring these states during the development process in order to gain a more precise picture of the current status of a development project could be an inspiration for VENUS to pay more attention to monitoring and steering the actual development process in addition to combining disciplinary practices into a methodology framework.

On the other hand, the SEMAT kernel does not have means to represent explicitly the social embedding of software systems, i.e. to introduce into the development process the abstract norms, obligations and rules of our society that often influence substantially the design and usage of software systems and thus determine the user acceptance to a very large degree. Clearly, it is an open and debatable question whether such concerns should be optional add-ons to the SEMAT kernel (extensions are explicitly foreseen in the SEMAT approach) or should be a first-level ingredient of the (so-called) essence of software engineering, as defined by SEMAT. From the perspective of VENUS we would argue for the latter.

Acknowledgement: We thank all members of the interdisciplinary Research Center for Information System Design (ITeG) at Kassel University for their contributions to VENUS, and Hesse's State Ministry of Higher Education, Research, and the Arts for funding the VENUS project as part of the LOEWE excellence program.

References

1. Baraki, H., Geihs, K., Hoffmann, A., Voigtmann, Ch., Kniewel, R., Macek, B.-E., Zirfas, J.: Towards Interdisciplinary Design Patterns for Ubiquitous Computing Applications, Technical Report, Kassel University Press, 2014 *(to appear)*

2. Baxter, G., Sommerville, I.: Socio-technical systems: From design methods to systems engineering, Interacting with Computers, 23(1), 4-17, 2010.
3. Behrenbruch, K., Kniewel, R., Hoberg, S., Schmidt, L.: Evaluationsmethoden im Kontext iterativer Gestaltungsmodelle für adaptive und auf Kooperation ausgerichtete Anwendungen. In: Mensch & Computer 2010: Workshop „Evaluation Adaptiver Systeme (EASYS)", Duisburg, 2010.
4. Bertagnolli S., Lisboa, M.: The FRIDA model. In Analysis of Aspect-Oriented Software (ECOOP 2003), July 2003.
5. Brown, D.M.: Communicating Design: Developing Web Site Documentation for Design and Planning, New Riders Press; 2 edition, 2010, ISBN: 0-3217-1246-3
6. Comes, D., Evers, C., Geihs, K., Hoffmann, A., Kniewel, R., Leimeister, J., Niemczyk, S., Roßnagel, A., Schmidt, L., Schulz, T., Söllner, M., and Witsch, A.: Designing Socio-technical Applications for Ubiquitous Computing - Results from a Multidisciplinary Case Study, Proc. Distributed Applications and Interoperable Systems (DAIS 2012), Springer (2012), 194-201.
7. David, K. Geihs, K. Leimeister, J. M. Roßnagel, A. Schmidt, L. Stumme, G. Wacker, A. (Editors): Socio-technical Design of Ubiquitous Computing Systems, Springer, Heidelberg, 2014 *(to appear)*
8. Fujitsu, Ivar Jacobson Int., Model Driven Solutions, SOFTEAM, UNAM: Essence – Kernel and Language for Software Engineering Methods, Revised Submission, 2012, http://semat.org/wp-content/uploads/2012/02/2012-11-01.pdf
9. Geihs, K., Leimeister, J.-M., Roßnagel, A., Schmidt, L.: On Socio-technical Enablers for Ubiquitous Computing Applications, 3rd Workshop on Enablers for Ubiquitous Computing and Smart Services (EUCASS 2012), at 2012 IEEE/IPSJ 12th Int. Symp. on Applications and the Internet (SAINT), July 2012
10. Geihs, K.; Niemczyk, S.; Roßnagel, A., Witsch, A.: On the socially aware development of self-adaptive ubiquitous computing applications. it-it 56, 1, p. 33-41, Oldenbourg, 2014
11. Hammer, V., Pordesch, U., Roßnagel, A.: Betriebliche Telefon- und ISDN-Anlagen rechtsgemäß gestaltet, Springer (Edition SEL-Stiftung), Berlin/Heidelberg, 1993.
12. Jacobson, I., Ng, P.-W., McMahon, P.E., Spence, I., Lidman, S.: The Essence of Software Engineering: The SEMAT Kernel, Communications of the ACM, Vol. 55, No. 12, December 2012.
13. Jacobson, I.: The Essence (Presentation Slides), http://semat.org/wp-content/uploads/2012/06/The-Essence-2012-05-30.pdf
14. Lukowicz, P. Pentland, A. Ferscha, A.: From Context Awareness to Socially Aware Computing, IEEE Pervasive Computing, vol. 11, no. 1, pp. 32–41, 2012.
15. Mouratidis, H. Giorgini, P. Manson G.: 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.
16. Narayanan, A., Vallor, S.: Why Software Engineering Courses Should Include Ethics Coverage, Communications of the ACM, Vol. 57 No. 3, Pages 23-25, 2014.
17. Object Management Group: OMG Unified Modeling Language (OMG UML), Infrastructure, v2.1.2., Technical Report, November 2007.
18. Pohl, K.: Requirements Engineering, Heidelberg: dPunkt Verlag GmbH, 2008, ISBN: 3-8986-4550-9.
19. Prado Leite, J.C.S., Yu, Y., Liu, L., Yu, E.S.K.: Mylopoulos, J., Quality-Based Software Reuse. Proc. 17th International Conference on Advanced Information Systems Engineering CAiSE '05, Springer LNCS, vol. 3520, pp. 535–550, 2005.
20. Roßnagel, A.: Rechtswissenschaftliche Technikfolgenforschung – Umrisse einer Forschungsdisziplin, Baden-Baden: Nomos, 1993.