Requirements Engineering for Software vs. Systems in General

Hermann Kaindl
Institute of Computer Technology
Vienna University of Technology
Vienna, Austria
kaindl@ict.tuwien.ac.at

Marko Jäntti
School of Computing
University of Eastern Finland
Kuopio, Finland
marko.jantti@uef.fi

Herwig Mannaert
Normalized Systems Institute
University of Antwerp
Antwerp,
Belgiumherwig.mannaert@ua.ac.be

Kazumi Nakamatsu
School of Human Science and Environment
University of Hyogo
Himeji, Japan
nakamatsu@shse.u-hyogo.ac.jp

Roland Rieke
Fraunhofer Institute for Secure Information Technology
Darmstadt, Germany
roland.rieke@sit.fraunhofer.de

Abstract—**Are there fundamental technical differences between requirements engineering for software vs. systems in general? It seems as though even functional requirements can mean something more general for a system including mechanical parts than for software alone. Quality requirements on safety deal with humans and their relationship with some real artifacts in their environment, so they cannot be dealt with by software alone. However, reliability of underlying software will be important in this context. While the internal structure of software will not normally be specified in its requirements, structure of a more general system may well be. These are just examples of what should be discussed.**

With regard to intelligent enterprises, there exist defined methodologies for enterprise modeling. Much as any other complex system, an enterprise may be better understood through modeling. Once an enterprise is better understood, it may be easier to make it intelligent. Whatever technical system is to be developed in an enterprise, it needs to fit into. By connecting enterprise modeling and requirements engineering, the likelihood of such a fit is increased. For software development, such connections have been worked out and are part of defined methodologies, some of them based on object-oriented modeling. Are they applicable to the development of general systems?

Keywords-requirements engineering; software; systems; enterprises

I. INTRODUCTION

The panel discusses whether there are fundamental technical differences between requirements engineering for software as opposed to requirements engineering for systems in general. Each panelist has his own position as stated below.

II. PANELISTS AND THEIR POSITIONS

A. *Marko Jäntti*

In order to identify differences between requirements engineering of software and requirements engineering of systems one should start by clarifying the relationships between the concepts 'software' and 'system'. We can use a term information system to define the system. Besides software, an information system covers the hardware, infrastructure and people that use the system. Thus, system requirements engineering can be seen as a broader concept than software requirements engineering. Unified Modeling Language that is a widely used modeling notation can be used for modeling software structure and behavior [1]. UML can also be used to describe the physical nodes of a system (deployment diagram).

Unfortunately, software and system requirements engineering do not fully satisfy the needs of today's IT world that is becoming more and more service-oriented. Thus, the third aspect of requirements engineering is service requirements engineering. Service requirements typically include most of the functional and non-functional requirements of software products but also address some service-specific requirements such as service availability and quality of IT service support [2].

B. *Herwig Mannaert*

Though an information system is a much broader concept than software, the software on itself can be seen as a system as well. What software systems and various types of systems in general, including systems with mechanical parts and even enterprises [3], have in common, is that they can be regarded as modular structures. While no single generally accepted definition is known, the concept is most commonly associated with the process of subdividing a system into several subsystems, which is said to result in a certain degree of complexity reduction and facilitate change by allowing

 ISBN: 978-1-61208-184-7

modifications at the level of a single subsystem instead of the whole system [6, 7]. In software systems, one should strive to pay as much attention to the modular structure as mechanical systems currently do.

When considering systems in general — software systems, organizational systems, etc. — both a functional and constructional perspective should be taken into account [6]. The functional perspective focuses on describing what a particular system or unit does or what its function is. The structural perspective on the other hand, concentrates on the composition and structure of the system, i.e. which subsystems are part of the system and their relations. Equivalently, one could regard the functional system view as a blackbox representation, and the constructional system view as a whitebox representation. By blackbox we mean that only the input and output of a system is revealed by means of an interface, describing the way how the system interacts with its environment. As such, the user of the system does not need to know any details about the content or the inner way of working of the system. The main issue with respect to this approach in software systems, is that modules often exhibit hidden coupling that is not explicitly defined in the interfaces. The evolution towards service-oriented computing is, amongst other things, addressing this issue.

What also distinguishes software systems and software requirements from their mechanical counterparts, is that they are subject to change. Requirements evolve during the development of software systems, and both the requirements and the actual system will continue to evolve during the system lifecycle. It has been shown in [4, 5] that it is all but trivial for software systems to cope with these evolving requirements, and that this leads to structure degradation. This would also be the case for mechanical systems, but they are not required to evolve during their lifecycle.

C. *Kazumi Nakamatsu*

If we formalize logical structures of systems whatever they are software or human like systems, requirements for the system could be easily treated and implemented, especially for functional ones. However, if a system includes human factors, it would be much more complicated to model such systems than just mechanical systems. In order to model any kinds of systems, whatever human factors are included or not, we have developed a paraconsistent logic program called Extended Vector Annotated Logic Program with Strong Negation (abbr. EVALPSN)[8], which can deal with not only inconsistency but also human like reasoning such as plausible reasoning and some modalities such as obligation. Moreover we have used it for modeling man-machine systems such as the safety verification system for railway interlocking in order to avoid train accidents caused by human error.

As a conclusion, generally speaking, the EVALPSN based modeling is fitter for modeling systems including a lot of human factors than just software.

D. *Roland Rieke*

Architecting novel dependable systems or systems of systems poses new challenges to the system design process [9]. Dependability and security analysis is growing in complexity with the increase in functionality, connectivity, and dynamics of the systems. The application of models is becoming standard practice, in order to tackle this complexity and get the dependability and security requirements right, as early as possible in the system design process. A modeling framework for the specification of security and reliability requirements has to consider not only the structure and functional dependencies of a system but also the possible behavior. Actions in a model can represent software, hardware or human behavior. One way to specify requirements is, to define specific constraints regarding sequences of actions, which should occur or must not occur in a system's behavior. Actions in the model represent an abstract view on actions of the real system, therefore it has to be ensured, that the abstraction does not hide critical behavior. The requirements analysis should also consider the behavior of an attacker, which can be different in comparison to, e.g., the Byzantine fault model. An attack to physical components, for instance, to cut a vehicle's brake has to be done physically on site and so it can only attack one physical unit at a time. However, a remote attack to the software of a vehicular communication system could affect all vehicles at once.

E. *Hermann Kaindl*

Are all types of requirements equally relevant for software and systems in general? How can software achieve its functions? Actually, it is “dead” unless run on some hardware, mostly some general-purpose electronic computer. Only the calculations or symbol manipulations of such a computer as programmed by a piece of software may lead through myriads of state changes, i.e., some (internal) behavior. The results of the calculations or symbol manipulations to a given input are the functions of the software.

Contrast this with a chair, a very simply mechanical system. It achieves its function to support someone when sitting on it without any state change but only through its physical structure (and certain constraints on it). So, a general system including mechanical parts may have different ways of achieving functions than a software system alone.

So, an important difference to me between requirements engineering for software vs. systems in general is that mechanical parts may achieve functions by their structure and may, therefore, give rise to important structural requirements.

REFERENCES

[1] M. Jäntti, T. Toroi, “UML-Based Testing,” Proc. of the 2nd Nordic Workshop on the Unified Modeling Language (NWUML 2004), Aug. 2004, pp. 33-44.

 ISBN: 978-1-61208-184-7

[2] Office of Government Commerce, The Official Introduction to the ITIL Service Lifecycle. The Stationary Office, UK, 2007.

[3] D. Campagnolo and A. Camuffo, “The concept of modularity within the management studies: a literature review”, International Journal of Management Reviews, vol. 12, no. 3, pp. 259-283, 2009.

[4] Mannaert Herwig, Verelst Jan, Ven Kris.- Towards evolvable software architectures based on systems theoretic stability, Software practice and experience - ISSN 0038-0644 - 42(2012), p. 89-116.

[5] Mannaert Herwig, Verelst Jan, Ven Kris.- The transformation of requirements into software primitives : studying evolvability based on systems theoretic stability, Science of computer programming - ISSN 0167-6423 - 76:12(2011), p. 1210-1222.

[6] G. M. Weinberg, An Introduction to General Systems Thinking. Wiley-Interscience, 1975.

[7] C. Y. Baldwin and K. B. Clark, Design Rules: The Power of Modularity. Cambridge, MA, USA: MIT Press, 2000.

[8] K. Nakamatsu, and Jair M. Abe, “The Development of Paraconsistent Annotated Logic Programs”, Int. J. Reasoning-based Intelligent Systems, vol. 1, pp. 92–102, June 2009.

[9] A. Avizienis, J.-C. Laprie, B. Randell, and C. E. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Trans. Dependable Sec. Comput., vol. 1, no. 1, pp. 1133, 2004.

 ISBN: 978-1-61208-184-7