Constructing POSE:
A Tool for Eliciting Quality Requirements

Vladimir A. Shekhovtsov[1], Roland Kaschek[2], Sergiy Zlatkin[2]

[1]Department of Computer-Aided Management Systems
National Technical University "KhPI", Kharkiv, Ukraine
shekvl@kpi.kharkov.ua

[2]Department of Information Systems,
Massey University, New Zealand
{R.H.Kaschek | S.Zlatkin}@massey.ac.nz

Abstract: Quality requirements elicitation for new computer applications rests traditionally on interviewing stakeholders. That makes getting these requirements right more complicated than is necessary because the anticipated users not necessarily are good at talking about the kind of computer aid and its qualities they would appreciate. We suggest bypassing that problem by constructing and using POSE (parameterized online simulation environment). Within this environment, anticipated users can explore ways of computer support for their work; it then makes it easier to talk about a potential computerized aid and the qualities thereof. Using POSE also simplifies obtaining quantified scenarios related to the various system qualities. As a result, this environment aids in getting the quality requirements right.

Key words: business simulation, business games, requirement elicitation

1 Introduction

Getting the requirements wrong in a software project is a guaranty for project failure. In the early history of computing in the 20th century, requirements were not considered as particularly problematic. They furthermore were conceptualized as functional requirements. Quality issues were largely ignored. The attempts to find a way out of the software crisis and the advent of the Web have significantly contributed to changing that. Perceived from a more up-to-date viewpoint the issue to focus at is software quality. That view suggests perceiving functionality as a prominent but not necessarily the only or most important quality aspect of a system under development (SUD). Obviously, customer satisfaction not only results from the right functionality implemented but also requires that functionality implemented the right way.

Taking a quality driven approach to software development implies four immediate problems: (1) defining software quality, (2) getting the quality requirements right, (3) using quality requirements as software process driver, and (4) choosing out of a number of admissible artifacts one of those with highest quality. This paper is about getting the quality requirements right. Nevertheless, we briefly mention that the problems (3) and (4) can be solved based on formal approaches such as the Analytic Hierarchy Process or other decision-making methods. See, for example [Kas06], for a more in-depth discussion of that.

For suggesting a solution of the problem of how to get the SUD qualities right we certainly need to discuss the concept of system quality and furthermore to explain for what reason getting the SUD quality requirements right is a problem at all. Our view of SUD quality concisely is fitness for use. Consequently the required quality varies with the intended use and thus over the projects. One therefore cannot simply always use the same given set of quality aspects for capturing SUD quality. Second, one cannot easily, or at all, forecast the intended usage processes as these often are a specific blending of the various stakeholders' wishes. Third, even if one can anticipate the to-be processes it remains difficult to anticipate their quantitative aspects.

We therefore suggest creating and using for quality requirements elicitation a simulation based software environment POSE (parameterized online simulation environment) in which users can try out sets of usage processes and chose some of their key performance characteristics. Using such a tool is likely to simplify significantly talking about the requirements for the SUD. Certainly, we will have to perform respective empirical studies once POSE is constructed. This paper is dedicated to drafting POSE architecture.

Paper outline. We first discuss system quality and quality requirements elicitation techniques. We provide the POSE usage model and use cases in section 3, the draft architecture in section 4, discuss related work in section 5 and conclude the paper in section 6.

2 Background Information

We start with discussing the concept of system quality. After that, we look at the quality requirements elicitation techniques. These approaches are close to POSE in purpose but different in implementation.

2.1 System quality

According to the Oxford Dictionary Online (see http://www.oed.com) the quality of something is understood as the "… nature, kind, or character" of that something and nowadays is "… restricted to cases in which there is comparison (expressed or implied) with other things of the same kind; hence, the degree or grade of excellence, etc. possessed by a thing." For saying more clearly what we mean by system quality we therefore must say under what view we are going to compare systems of the same kind. Our respective view is the one of fitness for use under stated or implied conditions made by intended system users.

We limit the focus of our work to creating software products and ignore services. Thus, it seems sustainable to reuse some of the knowledge about product quality. We draw here in particular from Bralla [Bra96]. According to [Bra96, p.121-122] one can approach fitness for use in at least three different ways. First, one can use a taxonomy approach. In that approach, one can benefit from Garvin's work about product quality [Gar87] that is conceptualized as a hierarchy of quality aspects. Such taxonomy enables localizing systems in a reference framework and thus making trade-offs for favoring one quality aspect over another. Second, there is Taguchi's approach [TWS04] in which quality is defined as the reciprocal of total cost, which is understood as cost sum over all system life-cycle phases. This approach enables making designs for achieving trade-offs between life-cycle phases. That software maintenance is an essential cost factor might make this approach very appealing for organizations with long-term investment strategies for software. Third, there is a scenario-based approach due to Phadke [Pha89]. The quality score is here the degree to which the product delivers the required performance each time it is used, under the specified usage conditions, throughout the anticipated lifetime, and without harmful side effects. Considering scenarios of product performance potentially contextualizes performance and thus simplifies assessing it.

We decided to follow the road paved by Garvin because we do not anticipate that data of sufficient quality can be provided to follow Taguchi's approach. Phadke's approach seems to require knowledge about the system usage that we cannot come up with at the beginning of our analysis.

Garvin has used eight quality aspects of a product: *performance*, *features*, *reliability*, *conformance*, *durability*, *serviceability*, *aesthetics*, and *perceived quality*. Bralla added to this list the aspects of *safety*, *environment friendliness*, *ergonomics*, and *upgradeability*. We consider that as still insufficient for a list of always reasonably usable quality aspects. Furthermore, the terminology used in software engineering is somewhat different. We join the quality aspects that we have got from Bralla with the ones found in [GJM04] and in [Bar95] and add some of our views to obtain the list of quality aspects shown in Table 1.

N	**QA Name**	**QA Definition**
1	**Fidelity**	the degree to which S meets stated technical requirements
1.1	*Interopera-bility*	the degree to which S can be made to interoperate with systems that it was not required to interoperate with
1.2	*Portability*	the degree of ease with which S can be deployed and made operational on a software or hardware platform for which it was not built
1.3	*Reusability*	the degree to which S can be used for a purpose different from the one stated when S was created
2	**Maintain-ability**	the degree to which S can be kept delivering the anticipated services after its initial release
2.1	*Reparability*	the degree to which defects S might have can be fixed
2.2	*Evolvability*	the degree to which one can adapt S to fit modified requirements
3	**Performanc e**	the degree to which the services delivered by S are delivered in a timely manner
3.1	*Latency*	the degree to which the services S delivers fit the stated response window between stimulus occurrence time and response occurrence time
3.2	*Throughput*	the degree to which the services S delivers fit the stated number of

		completed responses per observation time interval
3.3	*Capacity*	the degree to which the services S provides can handle the specified load under the stated conditions
4	**Safety**	the degree to which the users of S can justifiably expect that S does not cause high risks to its environment
5	**Security**	the degree to which S provides its services only to authorized users and protects their intellectual assets maintained within S
5.1	*Availability*	the degree to which the stated resources are actually accessible in the stated way to respectively authorized users under stated conditions
5.2	*Integrity*	the degree to which the intellectual resources maintained within S can be changed only by users authorized for that
5.3	*Privacy*	the degree the intellectual resources maintained within S can be accessed only by users authorized for that
6	**Usability**	the degree to which it is easy to use S for its anticipated users under the stated conditions
6.1	*Understand-ability*	the degree to which for the intended users it is easy to figure out what services S provides, how to identify, invoke, and use them
6.2	*Memorabilit y*	the degree to which it is easy to memorize the services provided by S, how to identify, invoke, and use them
6.3	*Ergonomics*	the degree to which using S is comfortable and does not degrade its user health
7	**Usage experience**	the degree to which using S under stated conditions pre-disposes its intended users positively towards using it again.[1]
8	**Value**	the degree to which S delivers its services as expected
8.1	*Availability*	the degree to which the services S provides are ready for use; often measured as *MTF* / (*MTF*+*MTR*); *MTF* is a mean time to failure, *MTR* – mean time to repair
8.2	*Correctness*	the degree to which S delivers services to its users that behave according to its functional specification
8.3	*Reliability*	the degree to which S continues to be operative after it first was made operational. It often is measured in terms of *MTF*
9	**Robustness**	the degree to which S' service provisioning is reasonable regardless of what unanticipated circumstances occur

Table 1. Quality aspects to be used with POSE

We restrict ourselves to external quality, i.e., the quality of an SUD S that a system user can perceive because that it seems to be relevant for POSE. We are, however, aware of that another list of quality aspects may work as well as long as POSE would be adjusted to it. It is also important to state that it is not possible to exhaust system quality understood as fitness for use by any fixed taxonomy because the use and fitness can change and the new quality aspects become needed.

We aim at using utility trees [KKC00] for specifying quantitative aspects of SUD performance for making quality requirements more meaningful and concrete. Utility trees are rooted trees of height up to three. Their root is a quality aspect. Its children nodes decompose it into lower level quality aspects. The leaves of utility trees are scenarios that represent quantified service provisioning requirements.

[1] Preece et al. list a number of sub-aspects of usage experience [Pre02, p. 18 - 20]. We do not aim at defining these here, as we do not feel sufficiently competent for that.

2.2 Quality requirements elicitation techniques

Stakeholder-involving techniques. These techniques help to elicitate the quality requirements by working with stakeholders using traditional techniques (interviews, brainstorming, checklists etc.) Usually this is done by structuring the quality requirements in some way and use this structured representation as an aid for stakeholders. Goal-oriented techniques [Chu00] classify the quality requirements according to the structured system goals. Dictionary-based techniques [CY04, SK05] use special lexicons or glossaries to help stakeholders to organize and categorize the requirements. Quality Attribute Workshops [Bar03] is an organizational framework aimed at identifying quality attributes for the given SUD (system qualities supposed to fulfill the requirements) by working out their case specific interpretations (scenarios); negotiated identification, characterization, and prioritization of these scenarios.

The problem with these approaches is that many of the stakeholders neither are used to nor trained in reasoning about quality requirements without having working experience regarding the targeted SUD.

Requirements discovery techniques. The requirements sources for these approaches are free-text or structured requirements specifications. One uses Natural Language Processing, Information Retrieval (IR), and similar techniques for obtaining the requirements from these documents in automated way. According to the Theme/Doc method [BC04] one processes requirements specifications in search for keywords that are used to generate diagrams for the design phase. In [Ros04] one uses IR-based techniques (e.g. regular expression matching) for finding the requirements related to the particular system quality. Following the approach in [Cle06] one defines the set of indicator terms (keywords), trains this set on the sample sentences (determining the weights indicating the relevance of the terms for the requirements categories), and classifies the document sentences into requirements categories based on the total relevance weight of every sentence's terms. The problem with these approaches is limited stakeholder participation in the retrieval process. In fact, after the elicitation process is completed, the requirements still need to be verified by stakeholders.

Domain-specific approaches. Some elaborated techniques exist for the particular classes of quality requirements. The TROPOS project [Bre04] supports some techniques and tools, such as security reference diagrams, security constraints etc., specifically aimed to eliciting security requirements from various sources. Misuse cases [Ale03] help to elicit security requirements by describing scenarios of improper use of the system. Other quality requirements addressed are performance requirements [Nix98], reliability requirements etc.

3 Using POSE

We first aim at describing a POSE usage model. From it, we then derive POSE use cases and later on invent a POSE architecture that enables implementing the use cases in an integrated system. We anticipate that there will be to be three different modes of using

POSE: (1) an adaptation mode for adapting the simulation environment to the case at hand; (2) an experimentation mode for trying out a number of different patterns of key parameter values; (3) an analysis mode for analyzing the experiments and formulating the elicited requirements. Two remarks are already obvious. First, POSE needs to maintain a session context so a user can pause a session and continue experimenting with it later for finding out the most appreciated collection of software components for everyday work. Second, one cannot necessarily assume that the POSE usage modes occur in temporal succession. A feedback from experimentation to adaptation might be a required form of control additionally to feed forward from adaptation to experimentation.

3.1 Basic idea

We consider an organization as a system that enacts a number of business processes. Thus, for simulating an organization all we need is to model its structure, the resources and humans it utilizes, as well as the business processes, supporting- and management processes. For the latter we reuse a software tool originating from the work on associative retrieval and reuse of business processes: the *process assembler* (PA) [ZK05]. PA stores process models encoded in various process modeling languages (PML) and enables mapping models from one PML into another provided the expressivity of the PML permits such mapping and suitable mapping drivers are registered. The PA provides a web service to external systems of business partners of the PA's owner. POSE is a respective example.

As for POSE, the communication process between the POSE and the PA should be as follows. The POSE generates a request for a process model, specifying the POSE internal process modeling language, and other parameters according to the PA's metadata. The PA retrieves the best matching models using its associative retrieval capacity; and, if retrieved model is encoded in a modeling language POSE cannot handle, maps this model into a suitable language, provided respective driver exists.

Given the mentioned models are available as well as load information, i.e. information regarding the probability distribution with which processes are triggered, the required processing time for tasks in these processes, and the resource utilization etc. one can in principle simulate the whole organization. What we have in mind is then to let a potential user of an SUD experience a simulation that incorporates key SUD usage parameters. We intend to admit the connection of servers in the simulation model with models of the software component that aids a human user to perform the request. That way we can provide a realistic usage experience of the SUD functionality on top of the statistical analysis that comes with the simulation model.

3.2 Adaptation mode

One of the critical tasks the POSE users need to solve in the adaptation mode is the initial definition of the scope of the intended simulation, i.e., defining what counts as the organization under scrutiny. This task requires an understanding of the business processes one wants to aid by the SUD under scrutiny. In the adaptation mode, one has then to work out the business processes of that organization. We do not go into detail regarding how to do that. We are sure to find respective ways later.

POSE users during adaptation mode provide all data, in initial form, that they need for conducting simulation experiments. Thus, one enters the structure of the organization including the roles of its staff and the load information in the adaptation mode. This process is called *POSE parameterization* it covers typical load data such as occurrence figures, arrival patterns, tool utilization, availability of staff and other resources, probability of disasters of various kind, and the severity thereof, etc.

3.3 Experimentation- and analysis mode

In this mode, the processes defined in the adaptation mode are simulated. On users' demand, models of SUD-components that are registered with POSE can be animated or tried out if they actually provide executable code. Redefinition of the parameter values entered during POSE parameterization is certainly possible. Additionally, POSE users can make comments on the perceived system performance. They furthermore can assess registered SUD-components in formalized ways. First, they can rank an SUD version on a scale [1, 10]. The scale value size corresponds here to that users' confidence in that version. Second, the users can accredit to each pair (V_1,V_2) of two SUD versions the predicates "much better(V_1,V_2)", "better(V_1,V_2)", "marginally better(V_1,V_2)", and "equal(V_1,V_2)".

In the analysis mode one analyses the requirements data obtained in the experimentation mode. POSE provides one of its end users, i.e., a requirements engineer a number of reports with pre-analyzed experimentation data. Each of these reports, of course, includes the version number of the involved model, processes, load characteristics, SUD-components, and organization structure respectively.

3.4 POSE user types

POSE experts perform the POSE adaptation. A POSE expert is a **business process expert** (PE) or an **SUD expert** (SE). A PE establishes processes of SUD usage. A PE is proficient in the business processes of the company and able to transfer their knowledge into the activities of the adaptation mode. A PE is usually not familiar with the intricacies of the SUD development; in most cases, they view the SUD as a "black box". An SE elaborates and creates SUD-models and SUD component prototypes and integrates these into POSE. We presuppose that qualified developers are perfectly suited for the SE role.

A **stakeholder** (BS) is one who is capable of figuring out the qualities of the SUD in a process of playing with POSE. Some stakeholders have an overview over the company's core business and understand how the SUD is related to it. We expect such stakeholders to be knowledgeable about intended SUD business usage processes and external SUD quality requirements such as performance or availability. Other stakeholders are IT professional responsible for the organization's IT infrastructure, i.e., activities such as system deployment, upgrading, and maintenance. They likely provide the requirements related to the qualities of the software process involving the SUD. Examples are security, maintainability, and upgradeability requirements.

A **system administrator** (SA) supervises POSE execution regularly. An SA is responsible for performing administrative duties such as backups, daily maintenance. They also are responsible for successful execution of the simulations and perform the roles of the simulation administrators (start simulations, instruct and guide first-time users (stakeholders), resolve conflicts, make announcements etc.) In addition, SA should be capable of parameterizing the simulations.

3.5 POSE use cases

In the following tables, we list the most important POSE use cases.

ID	Actor	Name	Basic Flow of Events
1	SA	Manage User	Add, delete, modify a user
2	PE	Manage Structure	Maintain an organizational structure (OS). Add, delete, or modify an organizational unit (OU), an OU-relationship, a staff member (STM), STM-assignation to OU
3	PE	Manage Process	Maintain a usage process (UP). Add, delete, or modify UP-components (e.g. notational parts of a BPMN diagram) or links between UP and OS. Define the UP's label (i.e., its "good", "bad", or "neutral" value).
4	PE	Identify Process	Maintain a UP-set. Add a UP to or delete a UP from the set. Change the UP-set's status (i.e. its "to-be" or "as-is" values).
5	PE	Manage Role	Maintain a user role. Add, delete, or modify a role, an artifact-related access right (AR), AR-assignation to role, or an AR characteristic.
6	SE	Manage Component	Maintain a version of a SUD component (SC) as its process (SCP). Add, delete, or modify SCP-components (see use case 3) or links between SCP, UP, and OS.
7	SE	Manage SP/ES	Maintain a version of a SUD component as an existing SUD prototype (SP) or an external system (EP).
8	PE SE	Manage Simulation Model	Maintain a simulation model (SM). Add, delete, modify, export, or import a SM, an (OS-, STM-, UP-set-, R)-assignation to SM

Table 2. Use cases for the adaptation mode

ID	Actor	Name	Basic Flow of Events
9	SA	Parameterize Artifact	Maintain a parameter value (PV) connected to an artifact. Add, delete, or modify a PV, and a PV-assignation to an artifact.
10	SA	Manage Stakeholder	Maintain a stakeholder (BS). Add, delete, or modify a BS, (U, R, or S)-assignation to a BS.
11	SA	Control Simulation	Control a S execution state. Start, stop, pause, or continue a S.
12	BS	Play with Simulation	Control a simulation session (SS). Start, stop, pause, continue a SS. Make decisions, perform actions, and input values via a SS interface.

13	BS	Assess Quality	Maintain an assessment result (ASR). Add an ASR via an assessment interface launched from a SS.
14	PE,SE, BS	Manage Report	Maintain a simulation report (SR). Add, delete, or modify a SR, a (S-, PV- or ASR-) assignation to a SR.

Table 3. Use cases for experimentation- and analysis mode

4 POSE Architecture

Since we expect POSE to have multiple concurrent users who flexibly choose between several alternative representations of the simulation model a model-view-controller architecture (see [Bus99]) for POSE appears as suitable. We depict a high-level view of that architecture in Fig.1. Since the diagram utilizes standard notation we do not provide a legend. In the sequel, we are ignoring the required versioning schema and incorporate it later into our architecture. For simplifying the concurrent use of POSE, we are going to provide Web access to it.

The architecture shows users of the four types defined above interacting with POSE. After SA has defined the users and initialized the **access control,** PE and SE in interaction with the **model builder**, create the **simulation model**. After that, the BS can utilize the MVC part of POSE. Note that the **controller** is in charge of feeding BS comments and assessments via the model into the requirements DB. The controller is also the one the BS asks for changing the perspective on the model, such as a change of the statistical performance parameters available via the **view**. The same goes for animation, simulation, or trying out software component models associated with servers in the simulation model. PE and SE use the **model builder** for working with the simulation artifacts in the adaptation mode. That component forms the data necessary to run the simulation. The **exchanger** is for import and export of POSE data. The **maintainer** is for SA to maintain POSE daily.

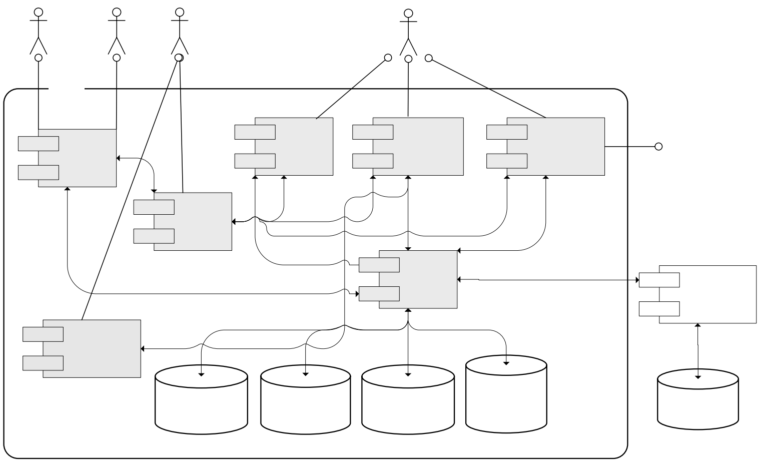

Figure 1: High-level view of POSE's architecture

The model builder maintains a number of artifact types corresponding to PE- or SE use cases listed in Table 2 (such as organization structure, organization unit, simulation user, user role, simulation model etc). That data after pre-processing is passed on to the data manager.

Architecture of the model component

The simulation model component contains the simulation model of the anticipated working situation in which an anticipated user works with POSE in such way that it is possible to elicit quality requirements via analysis of this work.

The key parts of the simulation model component are enactor, analyzer, and data manager. The **enactor** is responsible for running the simulations in the experimentation mode. It receives the simulation data from the model builder via data manager and performs the interactive parameterized simulation execution with quality assessments. All the simulation execution data is stored into the database via data manager. The enactor is also responsible for invoking animation, trial execution of registered SUD-component, and the respective registration.

The **analyzer** is for managing the simulation execution reports in the analysis mode. It receives the execution data from the enactor via the data manager and creates reports of different kind based on this data.

The **data manager** is responsible for managing all the data necessary to run the simulations. To obtain that data, it communicates with the model builder in a process of editing the artifacts data, collects edited artifact data, sends the retrieval- and update requests to the PA, converts the received model data into executable form, and supplements this data with the simulations parameters and assessment information. It then transfers the data to the enactor for simulation, obtains the simulation results back, add these results to the data and transfers it into the analyzer. It also controls the databases in POSE as well as the activity of the exchanger.

The enactor obtains the data from the data manager and executes it. With the load data, the organization structure and the processes retrieved from the PA the enactor creates a simulation model. Creating this model is relatively straightforward as soon as the process actors, i.e. the resources that drive the process are identified. The enactor communicates with the controller for attaching BS comments regarding the SUD experience made in simulations to those SUD components that are chiefly affected.

5 Related Work

Work has been published regarding the development and execution of simulations of a SUD for eliciting or verifying the system requirements. The most widely known projects of this kind are Statemate [Har90], SCR Toolkit [Hei05], approaches by Seybold, Meyer, and Glinz [SMG05], and Egyed [Egy04]. There are also several proprietary commercial tools with similar purpose. IRise [iRi07] and Simunication are respective examples. These tools, as well as POSE is intended to do, execute interactive simulations allowing stakeholders to participate.

POSE differs from these works in a number of ways. First, the published research approaches interactively simulate a SUD but only as a standalone system. Integrating into these simulations the SUD usage processes is not well supported. For example, using SCL (Statemate's control language), one can only develop usage scenario manually for every simulation run. Describing the usage processes involving the SUD in this situation looks like programming business processes in general-purpose programming language not tailored towards specifying processes involving human actors. To the contrary, POSE aims at simulation of the entire environment for the SUD using BPM techniques. The roles of both the user and the SUD are completely specified in this environment.

The approach of [Egy04] is closer to POSE in that it pays stronger attention to the SUD usage modeling. The SUD itself is represented with a software component; some software process activities related to this component are modeled (component replacement, component upgrading etc.) However, as for other approaches, these activities are not described as components of SUD usage processes.

Additionally, all these approaches differ from POSE in their purpose. We do not know about any approach using simulation to elicitate the required system qualities based on user experience of working with the simulation. For Statemate and SCR, the aim of simulation is validation of the requirements model built prior to simulation. For [Egy04], the goal is to verify the structure of the component and its interactions with environment. The output for these approaches is the validated functional requirements model of the system, no system quality assessments are supposed to be obtained.

The goal of the tools like iRise and Simunication is to allow non-programmers to build executable models simulating external behavior of the system and execute these models to receive feedback concerning the quality of the simulated interface and the required functionality as seen via this interface. In fact, they allow capturing such requirements as usability and user-friendliness, but no special attention to them is paid, so stakeholder assessments of such qualities can be captured as freeform user notes only.

To the contrary, POSE simulations specifically aim at gathering user assessments of the simulated SUD's quality. POSE uses functional specification only as its input, its output is an initial version of user assessments of desired system quality. At least for complex SUDs a further analysis of these initial assessments should be carried out with established methods.

6 Conclusions and future work

We are going to implement a first prototype of POSE. After that, we are aiming at validating it in a requirements elicitation case study. We aim at showing that POSE at least in specific cases is more effective and efficient in eliciting quality requirements than are traditional and other competing methods.

Bibliography

[Ale03] Alexander, I. Misuse Cases: Use Cases with Hostile Intent. IEEE Software, Vol. 20 (1), Jan/Feb 2003: 58-66.

[Bar95] Barbacci, M.; Klein, M.; Longstaff, T.; Weinstock, C. Software quality attributes. Technical Report CMU/SEI-95-TR-021, CMU, 1995.

[Bar03] Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J., et al. Quality Attribute Workshops (QAWs), Third Edition. Technical Report CMU/SEI-2003-TR-016, CMU, 2003.

[BC04] Baniassad, E.; Clarke, S. Finding Aspects in Requirements with Theme/Doc. In: Proceedings of Early Aspects 2004, Lancaster, UK, 2004.

[Bra96] Bralla, J. Design for Excellence. McGraw-Hill, Inc.: New York et al. 1996.

[Bre04] Bresciani, P.; Giorgini, P.; Giunchiglia, F.; Mylopoulos, J. et al. TROPOS: An Agent-Oriented Software Development Methodology. In: Journal of Autonomous Agents and Multi-Agent Systems, Vol.8 (3), 2004, pp. 203–236.

[Bus99] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M. Pattern-Oriented Software Architecture: a System of Patterns. John Wiley & Sons. Chichester et al. 1999.

[Chu00] Chung, L.; Nixon, B.; Yu, E.; Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Boston: Kluwer Academic Publishers, 2000.

[Cle06] Cleland-Huang, J; Settimi, R.; Zou, X.; Solc, P. The Detection and Classification of Non-Functional Requirements with Application to Early Aspects. In: Proceedings of RE'06 Conference, 2006.

[CY04] Cysneiros, L.M.; Yu, E. Non-Functional Requirements Elicitation. In: Perspectives on Software Requirements. Kluwer Academic Publishers. 2004, pp. 115-138.

[Egy04] Egyed, A. Dynamic Deployment of Executing and Simulating Software Components. In: Proc. 2nd IFIP/ACM Working Conf. on Component Deployment, 2004, pp. 113-128

[Gar87] Garvin, D.A. Competing on the Eight Dimensions of Quality, In: Harvard Business Review, Vol. 65 (6), 1987, pp. 101-109.

[GJM04] Ghezzi, C.; Jazayeri, M.; Mandrioli, D. Software Qualities and Principles. Chapter 101 of Allen Tucker. Computer Science Handbook. Chapman & Hall/CRC. 2nd. ed. 2004.

[Har90] Harel, D.; Lachover, H.; Naamad, A.; Pnueli, A. et al. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. In: IEEE Transactions on Software Engineering, Vol. 16, April 1990, pp. 403-414.

[Hei05] Heitmeyer, C.; Archer, M.; Bharadwaj, R.; Jeffords, R. Tools for constructing requirements specifications. In: Comput Syst Sci & Eng. Vol.1, 2005, pp. 19-35.

[Heu04] Heufler, G. Design Basics: from Ideas to Products. Niggli Verlag AG: Sulgen. 2004.

[iRi07] iRise tool home page. URL: http://www.irise.com, accessed on 22.02.2007

[Kas06] Kaschek, R.; Pavlov, R.; Shekhovtsov, V.; Zlatkin, S.: Towards Selecting among Business Process Modeling Methodologies. In: Proc. of BIS 2006, Klagenfurt, Austria, 2006.

[KKC00] Kazman, R., Klein, M., Clements, P. ATAM: Method for architecture evaluation. Technical Report CMU/SEI-2000-TR-004, Software Engineering Institute, CMU, 2000.

[Nix98] Nixon, B. Managing Performance Requirements for Information Systems. In: Proceedings of WOSP'98, Santa Fe, NM., p. 131-144.

[Pha89] Phadke, M.S. Quality Engineering Using Robust Design, Prentice Hall, 1989.

[Pre02] Preece, J., Rogers, Y., Sharp, H. Interaction design: beyond human-computer interaction. John Wiley & Sons, Inc. 2002.

[Ros04] Rosenhainer, L. Identifying Crosscutting Concerns in Requirements Specifications, In:

Workshop on Early Aspects, Vancouver, Canada, Oct. 2004.

[SMG05] Seybold, C.; Meier, S.; Glinz, M. Simulation-Based Validation and Defect Localization for Evolving, Semi-Formal Requirements Models. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference, 2005, pp. 408-417.

[SK05] Shekhovtsov, V.A.; Kostanyan, A.V.: Aspectual Predesign. In: Information Systems Technology and its Applications - ISTA'2005. LNI P-63, GI-Edition, 2005, pp. 216-226.

[TCW04] Taguchi, G.; Chowdhury, S.; Wu, Y. Taguchi's Quality Engineering Handbook. Wiley, 2004.

[ZK05] Zlatkin, S.; Kaschek, R.: Towards Amplifying Business Process Reuse. In: Perspectives in Conceptual Modeling: ER 2005 Workshops, LNCS 3770, pp.364-374, Springer, 2005.