Stakeholder Discovery and Classification Based on Systems Science Principles

Otto Preiss
Department of Information Technologies
ABB Corporate Research Ltd
5405 Dättwil, Switzerland
otto.preiss@ch.abb.com

Alain Wegmann
Department of Computer Science
Swiss Federal Institute of Technology
1015 Lausanne, Switzerland
alain.wegmann@epfl.ch

Abstract

It is the goal of our research work to elaborate on improvements to the software development methods so that quality attributes can be handled more systematically. By quality attributes we mean the large group of typically systemic properties of a software system, such as availability, security, etc., but also reusability, maintainability and many more. We define quality attributes as stakeholder-centric conditions on the behavior or structure of a system. The importance of the notion of a stakeholder cannot surprise, but the lack of a general theory on how to define and identify the relevant set of stakeholders does. Drawing from systems theory we claim that four basic, generic types of stakeholders are sufficient to be able to derive a specialized set of stakeholders for any considered system and domain of inquiry. It is only when we understand the generic concepts and principles behind quality properties of systems, that we can properly derive methods and build tools to cope with them.

1. Introduction

The increasing popularity of compositional means to develop software, driven by component-based software engineering, or Web service composition, has raised the priority of all extra-functional aspects of software systems. The absence of fully specified reusable assets together with the composition of black-box (possibly commercial-of-the-shelf) components and services, results in less confidence in the proper prediction of the overall system behavior, in particular the different qualities of behavior. However, to reason about properties a posteriori immediately calls for their reference, i.e. for a wellness grading of these properties. It is the stakeholder-derived requirements that are at the root of the relevant properties and their achievement criteria. In the context of a software system, quality attributes refer to the large group of properties, sometimes referred to as "Ilities" [1], which are either discernable at system runtime (such as dependability, usability, safety, security, consistency) or observable over the product lifecycle (such as extendibility, evolvability, reusability, etc.). Since quality attributes are the motivation but not the primary subject of discussion in this paper, we refer to [2] for a more exhaustive list of software related quality attributes as well as references to quality attribute related work.

While there is agreement neither on the set of quality attributes nor on a classification of them there is consensus that the relevant attributes are dependent on the various stakeholders, who represent parties that have stakes on the behavior of the system or the way the system is being built. Ramesh [3] states that "high-end [requirements] traceability users" recognize stakeholder traceability as one of the most important aspects in their software process improvement programs. Unfortunately, the identification and characterization of the relevant set of stakeholders for a certain system at a certain moment in time is largely unclear and thus done in an ad hoc way, supported only by experience and intuition. We believe that by adopting some fundamental principles of systems science, we are able to provide a generic, scientifically recognized, basis that can aid the stakeholder discovery and classification.

2. Concepts and Principles Derived from Systems Science

Systems science, being part of the systems theory framework, is concerned with processes of complex systems. It studies both the commonalties of all complex systems and the models to describe them. It claims that the same principles and concepts are applicable to the different disciplines, such as physics, biology, technology, etc. Systems may be living, nonliving, or mixed living and nonliving [4]. A typical software system, being a man-machine system, falls into the latter category. As opposed to classical science, system science is not a reductionism approach, but promotes the theory, that the behavior of a system cannot be predicted by looking at the individual

0-7695-1287-9/01 $17.00 © 2001 IEEE

parts only. For us, the value of systems science lies in its generic and therefore trans-disciplinary means to study and model systems.

In order to develop a conceptual scheme of a complex system, i.e. a system model, we must identify and describe the set of concepts and their relationships, from which we can then construct the general system principles. Since a system is just the perceived behavior of reality, and a system model the "...organized description of an existing or designed future system"[5], we imply that *a model of a future system is in fact only an organized description of the perception of the future reality*. This perception is, of course, dependent on the viewer, whom we represent by the notion of a stakeholder. A stakeholder is a person entrusted with the stakes of bettors [6], i.e. someone whose profession entails to be concerned with the outcome of a system action. The idea of a viewer-specific perception is in line with the theory of uncertainty, which has helped us to learn that "the observer cannot be separated from what is observed" [5].

With respect to quality, which is defined as the "grade of excellence" [6], we infer that *Quality attributes of a system describe the grade of excellence of a perceived (future) behavior of reality determined by a stakeholder.*

From this we conclude that the notion of stakeholder is central to capturing the entirety of qualities of a system, and especially, to define the quality requirements on a yet to be designed system. But how can we make sure that we identify all the relevant stakeholders for this undertaking? By adopting the systemic view of systems theory, we are guided by a framework that theoretically guarantees to identify the complete set of relevant, abstract concepts and thus also of all the stakeholders. Only if we have this, can we successfully design tools to support engineering and traceability with stakeholder-centric quality attributes.

The following fundamental definitions and principles of systems science and systemic modeling are influencing our work:

1. "System means a configuration of parts connected and joined together by a web of relationships." [5]
2. There exist two types of systems: Natural systems (e.g., living systems) and designed systems (e.g., manmade artifacts, human activity systems).
3. A complex system is one that cannot be divided into an independent set of sub-systems and that is open (i.e., it is always interacting with other systems in a dynamically changing environment). The relevant environment at any moment in time is comprised of the set of other systems, which, together with the system under consideration, makes up the suprasystem [7]. Classical science, on the contrary, views systems as essentially being closed with very limited and highly controllable interactions with the environment.
4. Studying systems involves both the study of the total system behavior ("the goal of the system") as well as the structure that performs this total behavior ("the means to achieve the goal").
5. The application of a systems view requires the consideration of the concepts and principles in a functional context, i.e., it requires selecting the type of system and the domain of inquiry.
6. A system attempts to organize itself in such a way that it serves the purpose of the individual parts as well as the purpose of the entire system (self-organization principle).

From the above rather philosophical statements we derive the following four, seemingly trivial, concrete interpretations and conclusions:

(a) *Software systems are complex, designed systems:* They either belong to the type of "human activity system", when the total software system is considered in its execution environment, or to "fabricated-engineered, physical systems (manmade artifacts)", when observed in isolation.

(b) *Two systems:* A software system cannot be modeled without considering the system it is embedded in (i.e., the environment). Consequently, there are always at least two systems to be considered: the suprasystem and the system under consideration.

(c) *Two viewpoints:* We must always explicitly analyze both the goals of a system and the means to achieve the goals.

(d) *Life-cycle based domains of inquiry*: The (software) system life-cycle lends itself to define the relevant domains of organizational inquiry.

This kind of systemic view, which is subsequently used as a classification basis, is depicted in Figure 1.

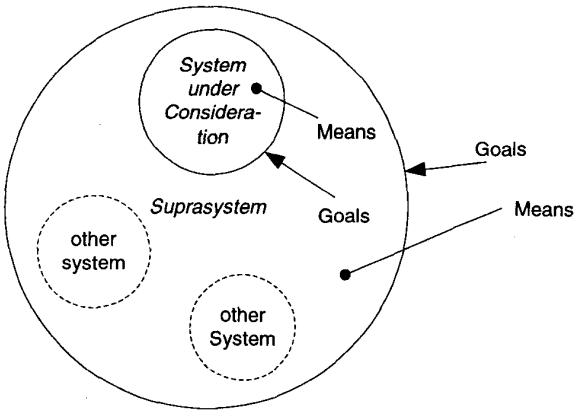

Figure 1. Two relevant system layers with "goals-means" viewpoints

Because principle 6 is not further explored in this paper but is relevant to our future research work, we

briefly show its concrete value by an example. If a company organization maps to a suprasystem and a business information system to the system under consideration (SuC), we can see that the self-organization principle has profound impact on our SuC. Because the interacting parts with the SuC (e.g. users, other systems, etc.) will strive for optimization of their personal goals as well as the overall goals of the suprasystem, the interaction pattern with our SuC are likely to change quite frequently. Supporting evidence for this generic observation is given by the never-ending streamlining activities of business processes (business reengineering)[1], which yield new or changing requirements on the business support systems we build. Secondly, a more subtle result of self-organization is the system user's behavior, which must be anticipated to change, too. A user will try to optimize her usage of the system, e.g., omit optional entries to save time, find (not intended) backdoors to circumvent inconvenient working procedures, or increasingly use shortcuts when being more proficient with the system. Hence, in order to provide sufficient flexibility for system changes, it is of utmost importance that we design our components that interface the other parts of the suprasystem, before any other, with as much anticipated and conceivable variability in interaction behavior as possible. To be able to do so, we need to increasingly emphasize the domain engineering aspects of systems development. Hence, the focus on early development phases is indeed justified.

3. Stakeholder Classification

Although the notion of stakeholder is found in many areas of software engineering we, the authors, do not know of any method or model to systematically discover and classify stakeholders. Requirements engineering and software architecture are two fields in which the concept of stakeholder is prominent. In requirements engineering, viewpoint-based requirement modeling uses stakeholders as viewpoint representatives (for a survey see [8]), methodologies exist for stakeholder requirements elicitation and negotiation [9] [10] [11], and scenario-based capturing of contextual knowledge [12] is expressed through different kinds of interactions between stakeholders and the system. However, they all implicitly assume that the relevant set of stakeholders was identified somehow. The discipline of software architecture uses the concept of stakeholder as a means to typify the audience interested in architectural concerns. Stakeholders are used to assess the quality of architecture [13] as well as to make the "customers" of an architect more concrete to derive the important aspects an architect has to consider [14]. Again, the systematic discovery of the set of relevant stakeholders is not defined.

[1] These activities are in turn caused by the changes of their suprasystem, i.e. the market environment, society, etc.

3.1. Stakeholder Classification Framework

In the following we propose a generic stakeholder classification scheme that is based on our derived, basic principles in section 2: *two systems, two viewpoints,* and *two domains of inquiry*. The life cycle based domains constrain our universe of discourse, i.e. they support the separation of concerns. They represent snapshots, and thus portray the system under consideration at certain moments in time. Life cycle based partitioning of domains of inquiry help to establish system boundaries and context, which limits the potential stakeholders to be considered.

While a finer granularity is always possible, we can limit ourselves to two life-cycle phases to make our point: creation and operation. More concretely:

1. *System development*; this includes the conception, the design of the envisioned system, and the implementation of the design. In essence, it provides all the developed artifacts of a development project.
2. *System operation*; this includes the execution of the running system in its real environment as well as the management of change to support evolution of the system.

The classification scheme has a generic layout as shown in Table 1. Section 3.2 gives a concrete, simple application of the framework by using the two above-mentioned domains of inquiry.

Table 1: Generic stakeholder classification layout

Domain of inquiry	Goal stakeholder for Suprasystem	Means stakeholder for Suprasystem
	Goal stakeholder for "System under Consideration" (SuC)	Means stakeholder for SuC

The informal definition of the generic stakeholders is the following:

- *Goal stakeholder for Suprasystem*; the type of stakeholder that is interested in the perceived behavior of the suprasystem only. It does not even know that our system under consideration is part of the suprasystem and it is also not interested in how the suprasystem achieves its behavior.

- *Means stakeholder for Suprasystem*; the type of stakeholder that is interested in the way the suprasystem achieves its behavior, i.e. its structure. Hence, these stakeholders would typically be concerned about some (or all) of the inner systems and their interactions.
- *Goal stakeholder for System under Consideration*: the type of stakeholder that is interested in the perceived behavior of the SuC. It does not know or care about the means by which this behavior is achieved.
- *Means stakeholder for SuC*: the type of stakeholder that is interested in the way the SuC achieves its behavior. I.e., such a stakeholder is interested in the internal structure of the SuC.

Note that these abstract stakeholder classes can be applied to any system. For instance, the system under consideration may be an individual software component that is a part of the application – the suprasystem.

3.2. An Example

In the following we present a simple example to illustrate the above proposed generic classification framework.

Let us assume we are a company producing e-commerce applications for online supermarkets.

The domains of inquiry define the suprasystem and the SuC. Our selection of these systems is influenced by business value chain considerations. A value chain is a sequence of actions, each adding value by transforming its input to value-added output, which in turn constitutes (part of) the input for the next action. For instance, an engineering company procures basic components and integrates them into a marketable system. The value-adding activity is the actual integration process, which is the core value-generating (and possibly protected) asset of this company. A second company might now procure such a system and produce an added value by employing the system to provide a service to their customers. This generic scenario is representative also for our e-commerce application.

In the "development" domain of inquiry, we define the suprasystem as being the development company with the development project being the SuC. The latter is producing the added value in the form of the development artifacts, of which a subset (say the e-commerce application executable) is then input to the supermarket to help create added value in their system. Theoretically, all stakeholders identified in Table 2 will constrain, directly or indirectly, the realization of the software system to be developed. Ideally, we would analyze and model all, stakeholder specific, perceptions of the system to derive qualitative requirements. Since this undertaking is almost impossible to do, it is easy to understand why current development projects consider a small subset of stakeholders and models only. It is only the stakeholders discovered for the SuC in the operation domain of inquiry, which current development methods usually identify in the analysis phase as actors for our system to be built (i.e., direct users, other systems). It becomes evident that these primary stakeholders directly relate to the quality attributes that are discernable at system runtime (performance, usability, etc.). However, one can notice that by investigating the secondary stakeholders, we are lead to all other quality attributes, such as maintainability, reusability, etc.

Table 2. Stakeholders of an e-commerce application for a supermarket

Domain of inquiry	*Informal description of system*	*Goal stakeholder*	*Means stakeholder*
Development	Suprasystem: Development company	Company board; company shareholders; technology/tool provider	Company line management; employees
	SuC: Development project	Company marketing/sales; other company products[1]	Project member (programmer, architect, etc.); project management; other company projects; QA/process staff; company maintenance/support crew
Operation	Suprasystem: Company running the e-commerce application	Shopper; Supermarket suppliers (goods, etc.); Supermarket board; Supermarket shareholders;	Supermarket company line management; employees
	SuC: E-commerce application executing in target environment	Supermarket system user (back office, warehouse workers); in-house system administrator; in-house data maintenance personnel; other supermarket computer systems and applications (e.g. ERP)	Supermarket IT department; IT manufacturer/products for e-commerce platforms; e-commerce application vendor's hot line and maintenance crew

4. Conclusion and Future Work

We have derived some generic principles for software systems analysis and modeling from systems science. Basically, we suggest modeling any system by (a) considering the system and the embedding system of which it is part of, (b) by separating two distinct views, the "goal-centric" view from the "means-to-achieve-the-goal" view, and (c) by defining the system under consideration and its boundaries with the help of life-cycle partitioning. Among others, these general principles can be used to classify stakeholders. Furthermore, by treating the software development project, i.e. the process that generates a system, also as a system (essentially a life-cycle based partitioning of the domain of inquiry), we are able to conceptualize quality attributes very generically. More concretely, we are able to apply the same principles to discover and analyze both types of quality attributes, those that are observable during software system runtime (performance, dependability, etc.) and those that are not (reusability, maintainability, testability, etc.).

It is the goal of our research work to elaborate on improvements to the software development methods so that quality attributes can be handled more systematically. Quality attributes are stakeholder-centric conditions on a system's behavior, i.e. the visible goal and its realization. Therefore, we found it important, as a first step, to find or develop a theory on how to discover stakeholders and relate to their views and objectives. More empirical studies need to be conducted to evaluate the presented framework that resulted from the theory. Because stakeholder discovery and classification on its own is of limited value to industry, we intend to evaluate this part as soon as we are ready to conduct case studies based on a framework that covers stakeholders, quality attributes, and can be related to current software development processes.

In agreement with systems science, we believe that for understanding systems and systems modeling the emphasis should be much more on interactions, i.e. the collaborative behavior, than on the individual parts. Therefore, the notion of use cases and collaborations are central to our conducts in the second step [2]. Collaborations are our first-class behavioral concepts for which we want to investigate the application of feature modeling approaches [11] [15]. Opposed to current practice, we intend to restrict features to quality attributes. The feature space would thus represent the possible quality properties that can be or shall be realized by a collaboration of entities. In other words, a quality feature model would be a declarative, formal specification of the qualities of a behavioral concept.

Let us finally remark that software systems are still modeled and built under the premise of classical science (see definition 3 in section 2). Hence, requirements are captured in this world-view, although for instance organizations that shall use these software solutions are best modeled in the systems science worldview. We believe that this is one of the reasons, why software systems can hardly ever cope with the evolution of organizations and thus become inadequate, simply because the context (the suprasystem) has changed.

References

[1] F. Manola, "Providing Systemic Properties (Ilities) and Quality of Service in Component-Based Systems," Object Services and Consulting, Inc., Technical Report, 1999.

[2] O. Preiss, A. Wegmann, and J. Wong, "On Quality Attribute Based Software Engineering," accepted at 27th Euromicro 2001, Warsaw, Poland, 2001.

[3] B. Ramesh, "Factors Influencing Requirements Traceability Practice," *Communications of the ACM*, vol. 41, pp. 37 - 44, 1998.

[4] J. G. Miller and J. L. Miller, "Applications of Living Systems Theory," 2001: International Society for the Systems Sciences (ISSS), 1997.

[5] B. Banathy, "A Taste of Systemics," The Primer Project, A Special Integration Group of the International Society for the Systems Sciences (ISSS), Web-document, 2001.

[6] Merriam-Webster, *Collegiate Dictionary*: Merriam-Webster Online, 2001.

[7] J. G. Miller, *Living Systems*: University Press of Colorado, 1995.

[8] I. Sommerville and P. Sawyer, "Viewpoints: principles, problems and a practical approach to requirements engineering," *Annals of Software Engineering*, vol. March, 1996.

[9] W. N. Robinson and S. Volkov, "A Meta-Model for Restructuring Stakeholder Requirements," presented at International Conference on Software Engineering (ICSE 97), Boston, 1997.

[10] B. W. Boehm, P. Bose, E. Horowitz, and M.-J. Lee, "Software Requirements as Negotiated Win Conditions," presented at International Conference on Requirements Engineering (ICRE), 1994.

[11] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*: Addison-Wesley, 2000.

[12] K. Pohl and P. Haumer, "Modeling Contextual Information about Scenarios," presented at Third International Workshop on Requirements Engineering: Foundation for Sofware Quality RESFQ, Barcelona, Spain, 1997.

[13] S. Bot, C.-H. Lung, and M. Farrell, "A Stakeholder-Centric Software Architecture Analysis Approach," presented at 2nd International Software Architecture Workshop (ISAW-2) as part of SIGSOFT '96, 1996.

[14] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 6 ed: Addison-Wesley, 1999.

[15] R. W. Krut, "Integrating 001 Tool Support into Feature-Oriented Domain Analysis Methodology," Software Engineering Institute, Pittsburgh, Technical Report CMU/SEI-93-TR-11, 1993.