Introducing object-oriented requirements engineering for automation systems

H. KAINDL OVE, SENIOR MEMBER IEEE

This article raises the question, if and how object-oriented requirements engineering can be applicable and useful for specifying automation systems. In order to ease the judgment of applicability, it introduces this author's approach to object-oriented requirements engineering in the form of a mini-tutorial. Another such approach dedicated to specifying real-time industrial automation systems is sketched as well, and a combination of ideas appears feasible.

Keywords: **requirements engineering; systems engineering; software engineering; automation system**

Einführung in objektorientiertes Requirements Engineering im Hinblick auf die Spezifikation von Automationssystemen.
Dieser Artikel stellt die Frage, ob bzw. wie objektorientiertes Requirements Engineering für das Spezifizieren von Automationssystemen anwendbar und nützlich sein kann. Um die Beantwortung zu erleichtern, führt er in den Ansatz dieses Autors für objektorientiertes Requirements Engineering ein. Ein anderer solcher Ansatz speziell für das Spezifizieren von industriellen Automationssystemen mit Realzeit-Anforderungen wird ebenfalls skizziert. Eine Kombination der Ideen erscheint möglich.

Schlüsselwörter: **Requirements Engineering; Systems Engineering; Software Engineering; Automationssystem**

1. Background and introduction

Requirements engineering (RE) is "the branch of systems engineering concerned with the real-world goals for, services provided by, and constraints on a large and complex software-intensive system. It is also concerned with the relationship of these factors to precise specifications of system behavior, and to their evolution over time and across system families" (*Nuseibeh, Easterbrook, 2000*).

Although this widely used "definition" of RE has its focus on *systems engineering,* most of the work done and reported in this field deals with requirements in the context of *software engineering.* After all, this quote is from proceedings in a series of the most prestigious conferences on software engineering.

Another community explicitly dedicated to systems engineering has the following entry in the glossary of its systems engineering handbook (lacking one on requirements *engineering*):

"Requirements Analysis. The determination of system specific characteristics based on analyses of customer needs, requirements, and objectives; missions; projected utilization environments for people, products, and processes; constraints; and measures of effectiveness. Requirements analysis assists the customers in refining their requirements in concert with defining functional and performance requirements for the system's primary life cycle functions. It is a key link in establishing achievable requirements that satisfy needs" (*INCOSE, 2000*).

When comparing these "definitions" the question arises, whether there are any fundamental differences when dealing with requirements for software or more general systems. In fact, the requirements specification must not determine yet the solution as implemented in software or in a machine. Still, if mechanical subsystems are involved like, e.g., in building complete satellites, obviously physical requirements will be involved that software alone cannot address. For such systems, there will also be a larger set of stakeholders involved than for software systems. Even for computer-based systems, it may often be obvious from the very beginning that new hardware will have to be built for a particular task. However, sometimes it may have to be decided later in the development whether some special hardware is to be developed or whether available computers "just" have to be programmed for a solution. So, we think that for computer-based systems, RE should not be biased in the one or the other direction prematurely.

However, it makes a difference whether some business functionality is needed or whether the task is creating an automation system, probably with strict requirements on performance and safety. These latter non-functional requirements typically involve different feasibility studies and more precise specifications, e.g., of real-time constraints.

In general, requirements are usually described in natural language in practice. Even long time ago, however, the use of *models* was propagated. Creating such models is a complex activity of abstracting information and knowledge from a particular domain in order to achieve a model containing the essentials from the perspective of the modelers and their given goals (*Kaindl, Carroll, 1999*). Such models are primarily functional in so-called *structured analysis,* while they are models in terms of objects in *object-oriented analysis.* Object-oriented (O-O) models have penetrated practice in recent years much more than any other approach to software engineering before. Still, at least for dealing with requirements their use should be combined with natural language. This argument was published independently both from a software engineering and from a systems engineering perspective (see, e.g., *Kaindl, 1997; Oliver, 1997,* respectively).

KAINDL, Hermann, Univ.-Prof. Dipl.-Ing. Dr. techn., Vienna University of Technology, Institute of Computer Technology, Gußhausstraße 27–29, A-1040 Vienna (E-Mail: kaindl@ict.tuwien.ac.at)

In this article, the question if and how object-oriented requirements engineering can be applicable and useful for specifying automation systems is addressed. In order to set the stage for addressing this question, major part of this article sketches the use of O-O modeling for requirements engineering. Unfortunately, the O-O community has a relatively narrow view of requirements (*Jacobson et al., 1999*), and the RE community based in software engineering does not appear to focus much on O-O approaches, although they are the ones most widely used in practice. The systems engineering community is currently discussing and trying to adopt O-O approaches (*Kaffenberger, 2003*). This work of the author tries to bridge the gaps, so the following mini-tutorial on basics in object-oriented requirements modeling explains it from this author's viewpoint.[1]

Note, that requirements engineering has a much wider scope, even in the author's previous work, which also dealt with the following issues:

- traceability (*Ebner, Kaindl 2002*),
- pre-traceability through use of hypermedia (*Kaindl, 2001b*),
- reuse and reusability of requirements, product lines (*Mannion et al., 1999 and 2000*).

These and many other important issues cannot be elaborated further in this article due to space limitations. The interested reader is referred to textbooks, e.g., (*Davis, 1993; Loucopoulos, Karakostas, 1995*), the proceedings of conference and symposium series organized by IEEE[2] and INCOSE[3], respectively, as well as related journals[4].

Current commercial RE tools are well suited for *managing* large amounts of requirements written in natural language, but not (yet) for really *engineering* requirements in general. Such tools for requirements management and traceability are, for example, Requirements Traceability and Management (RTM, www.chipware.com/), Dynamic Object-Oriented Requirements System (DOORS, www.telelogic.com/), and RequisitePro (www.rational.com/).

The remainder of this article is organized as follows. First, the mini-tutorial on object-oriented RE is presented. Then its applicability to specifying automation systems is discussed. Finally, conclusions are drawn and future applications in the context of AUTCOMS are envisioned.

2. Mini-tutorial on object-oriented RE

Let us upfront provide a schematic system overview in order to discuss several notions of "system". Figure 1 illustrates this overview. We are convinced that any real *system to be built* (machine) will not "live" in a vacuum after deployment, but rather in an *application domain*. Human *users* will be involved, whether more directly or indirectly,. These people have tasks to perform and goals in their minds. Some of these tasks should be more easily performed and some of these goals more easily achieved after deployment. This should be the case after deployment of automation systems as well.

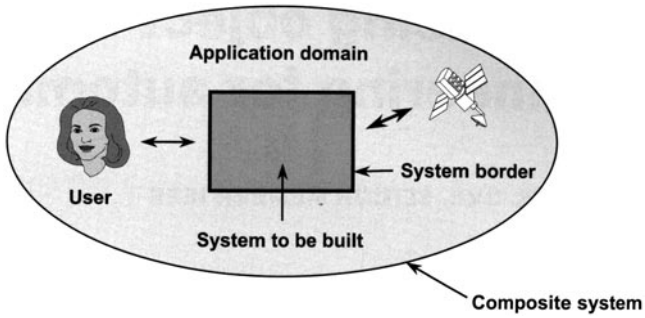

Fig. 1. System overview

Now let us have a more technical view. In addition to humans, technical entities may be in the application domain, e.g., satellites (much as in previous studies of this author for the European Space Agency). Whatever the system to be built may finally turn out to be (software alone or additional hardware, etc.), it will have a *system border* (sometimes also called automation boundary). This border should be clear and properly specified at the end of the RE effort, at the latest. However, it is often not predetermined from the outset. In fact, it may be one of the more intricate issues to define it. This definition includes task assignment, i.e., who will do what. Some tasks may still have to be performed by humans or other artifacts, and it should be clear which ones. So, there will actually be a more comprehensive overall system than just the one to be built. It is often called *composite system*.

This system overview should set the stage for this mini-tutorial. However, we prefer to use another running example than satellites: the ATM (automated teller machine) domain. We can assume familiarity with ATMs from their daily use. Unfortunately, for this reason it is difficult to imagine that we are working on requirements as though ATMs have not yet been existing. For the same reason, however, this example is useful for the purposes of this article, since this familiarity helps us to explain the notions and concepts discussed. In addition, this is the usual example in the O-O literature. So, it should be clear that the treatment of this domain here is naïve and should only serve demonstration purposes. Still, its relation to automation in contrast to pure software systems may help in the context of this article.

Figure 2 shows a very simple *domain model* (i.e., a model of the domain) before ATMs were deployed. This model is represented as a UML (Unified Modeling Language) *class diagram*.[5] This kind of diagram has been historically influenced by notations for entity-relationship modeling. However, the diagram in Fig. 2 does *not* specify an information model or even a data model of entities as represented inside some software system. (This is a major point of (*Kaindl, 1999*).) Rather, it models that there are several kinds of objects (*object classes*) in the domain, both physical objects (like `Cash notes`) and abstract objects (like `Cashier transaction`). In addition, it models a few properties of such objects in terms of so-called *attributes* (like `name` and `address` of `Customer`). The lines in between pairs of object classes signify that object instances (e.g., concrete cashier transactions and cashier stations) are related with each other (e.g., a specific cashier station is the handler of a specific transaction that happened at a particular date and time). This kind of relationship is called *association*. Such associations

[1] It is a written account of part of tutorials previously held by this author at various conferences: RE-02 (IEEE) in September, 2002, see www.re02.org/program/tutorials/t05.html; HICSS-36 (IEEE) in January, 2003, see www.hicss.hawaii.edu/HICSS36/tutorials.htm#Reconciling%20Business%20Modeling%20and%20Requirements%20with%20Object-Oriented%20Software%20Development; INCOSE-03 in June, 2003, see www.incose.org/symp2003/tutorials.htm#h03/.

[2] IEEE International Requirements Engineering Conferences, for the upcoming conference see www.re04.org/.

[3] Annual Symposia of the International Council on Systems Engineering, for the upcoming symposium see www.incose.org/symp2004/.

[4] Journals named *Requirements Engineering*, see rej.co.umist.ac.uk/, and *Systems Engineering*, see www3.interscience.wiley.com/cgi-bin/jhome/39084/.

[5] The specification of UML is available at the time of this writing at www.omg.org.

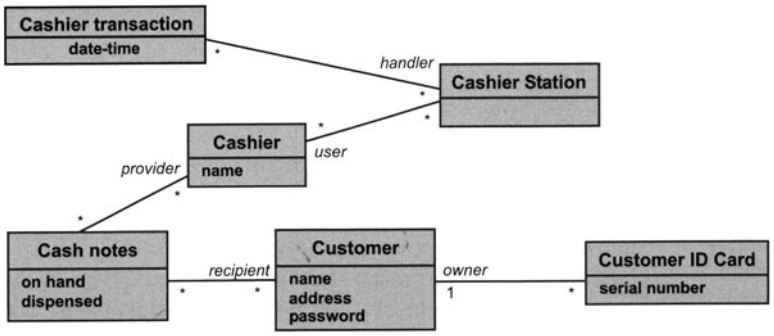

Fig. 2. Class diagram representing domain model as is

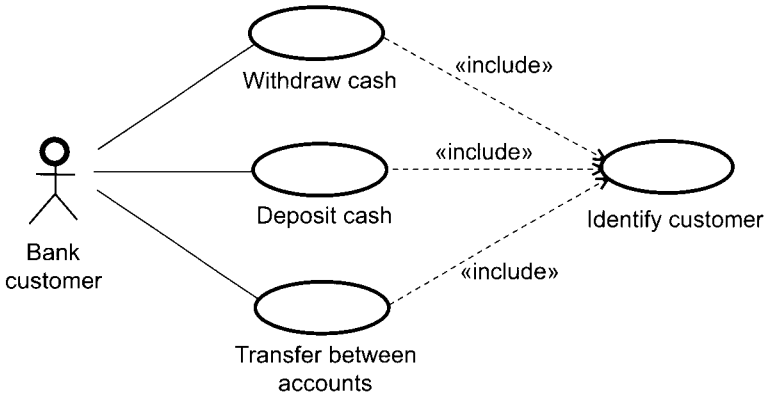

Fig. 3. Use-case diagram

may have cardinalities attached for indicating the numbers of instances allowed in such an association, but these are beyond the scope of this article. The primary purposes of such a model are to get a better understanding of the application domain serving also a better communication among stakeholders, and consequently a better understanding and specification of the requirements.

Such a class diagram shows a static view. The one in Fig. 2 represents a static model of the domain *as is* (as assumed to be a "world" without ATMs yet). In contrast, the diagram in Fig. 3 shows a so-called *use-case model,* indicating functionality that is envisioned for a "world" *with* ATMs (more precisely ATMs like the ones in the U.S., which provide additional functionality to cash withdrawal). We may view this as a functional decomposition, as a special one from the perspective of usage. While use cases (shown as ellipses) may have a variety of relations between them, this example shows just the `include` relationship, which serves factoring out common functionality (like customer identification in the use case `Identify Customer`).

Use cases are actually not "object-oriented", but they are a major approach for today's O-O modeling. They can be viewed as classes of usage *scenarios*, which are often defined as sequences of actions aimed at accomplishing some goal. An example for the use case `Withdraw Cash` is a scenario partly shown in Table 1. It describes a specific thread through the more general use case. UML provides special diagrammatic notation for a more formal representation of such scenarios, but this is beyond the scope of this article.

Even without connection to O-O, scenarios like these – but also with many variations – have been studied and used in software engineering and human-computer interaction as well. We consider this a healthy sign for this approach. In fact, also cognitive science provides support for it. The key advantage of such concrete usage scenarios over abstract specifications is that humans can both create and understand them more easily. However, it should be clear that normally no reasonably sized set of such scenarios can really serve as a complete specification.

Table 1. Part of scenario description

Customer	ATM
...	...
The customer enters the amount of money.	
	The ATM shall check the availability of this amount. If the amount is OK, the ATM shall provide the cash and request to take it. *By-function:* check amount, cash provision
The customer takes the cash.	
...	...

What makes the scenario description in Table 1 unique is the *By-function* attachment to the actions of the ATM. (It was introduced in (*Kaindl, 1997*) and defined and used more formally in (*Kaindl, 2001a*).) Intuitively, it means that performing these actions in the envisioned scenario shall be possible after deployment of the ATM *by* its *functions* as attached. Since these functions are required and not yet available, we view them as functional requirements. These may be specified by whatever means, but at least a concise and precise statement in natural language should be given, much as in traditional requirements specifications in good practice.

These attachments can serve as an integration of functional and scenario-based approaches. This is important to emphasize, since in current practice the application of the relatively modern scenario-based approaches is often thought to be incompatible with the traditional way of stating requirements.

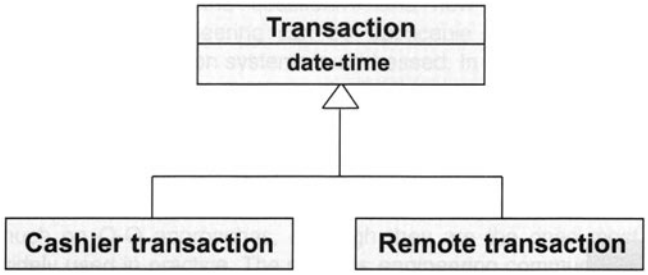

Fig. 4. Generalization in class diagram

Using this integration, however, both approaches can "live" together, leading to an overall advantage.

Even a certain way of functional decomposition using scenarios is possible in this way (see also (*Kaindl et al., 1998*) for a practical example). Use cases – with pre- and post-conditions – can define functional requirements for the composite system. The scenarios show how these functions shall be achievable, and the functions attached to the actions are functional requirements of the system to be built.

We think that a representation of the system to be built (an ATM class in our running example) should also be included in a static model *to be.* (This is also a major point of *Kaindl, 1999.*) Such a model can be represented through a class diagram like the one shown in Fig. 2. Actually, a model of the domain *as is* like in Fig. 2 need not always be built in practice. This could require too much effort. Building directly a to-be model, however, involves more cognitive overhead, since both observation and envisioning are involved more or less concurrently. For our running example, a to-be model should include also the new class `Remote transaction`. In fact, *generalization* should lead here to the class of a `Transaction` as well. Figure 4 illustrates this part of such a model. Note that the attribute `date-time` common to the more specific classes is factored out here. They are said to *inherit* it from the more general class. This inheritance is a key mechanism of information sharing in O-O approaches.

So, objects in domain models should represent objects from an application domain, while objects in design models represent abstractions of software objects (*Kaindl, 1999*). However, what about original requirements as provided by the prospective users of a system to be built? It can be useful to think of them as "objects" as well, much as when using a commercially available tool for requirements management like DOORS. The notion of an object in these systems is, however, not really the one of O-O approaches.

Our approach as published in (*Kaindl, 1997*) views requirements directly as *first-class* objects. A first-class object is an object that has all the features usually assumed in object-oriented approaches. I.e., there is no difference between objects in the domain model and requirements objects with respect to the representation in our approach. Requirements objects can be organized in generalization hierarchies that reflect different kinds of requirements. They can be related to both other requirements objects and domain objects, and they can have attributes. So, while requirements are not inherently "object-oriented", they can be usefully represented and organized according to object-oriented ideas.

As indicated above, any real system to be built (machine) will not "live" in a vacuum after deployment, but rather in an application domain. So, when having both the requirements and the domain modeled uniformly, these models should be interlinked, too. For this purpose, we defined a special association named *Statement-about,* since requirements will typically make statements about something in the domain. This is a major innovative aspect of the *metamodel* for requirements engineering that we defined in (*Kaindl, 1997*). (A metamodel shows how the corresponding parts of a model should look like.) A more comprehensive metamodel covering software design and implementation as well can be found in (*Ebner, Kaindl 2002*).

Let us show here in Fig. 5 the part of the metamodel which classifies requirements in `Envisioned Scenarios`, `Functional Requirements` and `Quality Requirements`. («`stereotype`» is a UML construct for defining metamodels.) Quality requirements (often called non-functional requirements) are constraints. As indicated above, e.g., safety requirements are important for automation systems and they are a special case of constraint on the system to be built. In addition, there can be constraints on the development process as well, such as specific tools or specific hardware to be used.

3. Applicability of object-oriented RE for automation systems

Is such an approach as sketched above applicable to specifying automation systems? In principle, we tend towards an affirmative answer, since this approach is very general. In fact, previous experience with it has shown that all kinds of requirements have been captured through it. However, certain aspects are represented more or less through natural language only. There-

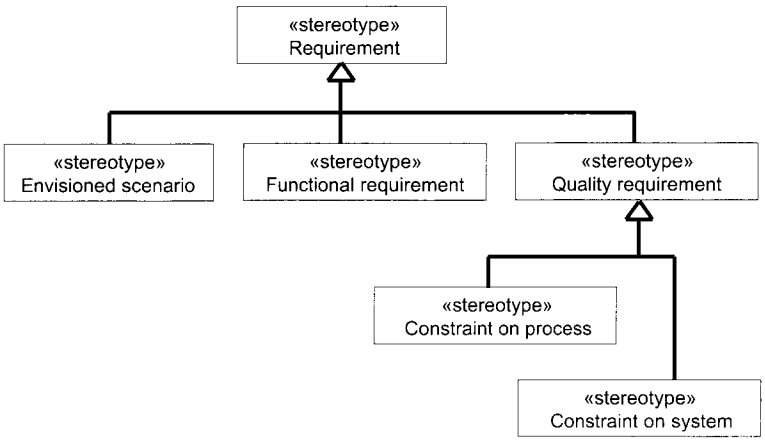

Fig. 5. Classification of requirements

fore, more formal treatment especially of time constraints may be needed for the specification of automation systems.

The principal applicability of an O-O approach to real-time industrial automation systems is also supported by the work of others, see (*Becker, Pereira, 2002*). This approach is even designed to support the whole development cycle of real-time industrial automation systems, including requirements. Major points of this work for addressing the question of this article are the following:

- This approach includes modeling of the technical plant to be automated in terms of objects (including physical domain elements).
- It includes simulation of the resulting model.
- There is some emphasis on modeling timing constraints.

Especially the latter two points may become important for certain applications to specifying automation systems and should, therefore, be investigated for inclusion into our own approach. On the other hand, our approach includes certain aspects that we could not find in (*Becker, Pereira, 2002*). We think that combination of the ideas appears feasible.

From a historical perspective, it is interesting to note that real-time extensions appear to be adopted later than other aspects not only in the context of O-O. The early work on O-O modeling in general was published much earlier than (*Becker, Pereira, 2002*). This development is somehow reminiscent of the history of *structured analysis.*

Structured analysis has its focus on functions and processes. The primary notation for the related functional decomposition are data-flow diagrams. For the real-time extensions, finite-state machines have been devised. This kind of automata have generally been the core of diagrammatic notations for real-time modeling. Note, that also *state charts* are (hierarchical) finite-state machines, and an O-O variant of them found their way into UML. So, from this perspective it should also be possible to learn from previous approaches for O-O RE in the realm of automation systems. As shown by (*Becker, Pereira, 2002*), objects can be useful for modeling such systems, even for simulating them.

4. Conclusion

In this article, the question if and how object-oriented requirements engineering can be applicable and useful for specifying automation systems has been addressed. While we are not yet in the position to provide a definite answer to this question, we sketched our O-O approach to RE and discussed it briefly together with another published approach that is specifically dedicated to real-time industrial automation. Based on this discussion, we conjecture that an affirmative answer may be given.

Still, more work is needed and especially real applications will have to provide empirical evidence. Especially for AUTCOMS, we suggest to focus on creating automation systems with a systems engineering view. It should include dedicated RE together with stakeholders, especially customers in projects for industry. Such applications also involve technology transfer, with all the difficulties and suggestions for solutions discussed in (*Kaindl et al., 2002*).

From a more scientific viewpoint, studies of potential differences between RE for systems and RE for software only, as well as a possible unification will be of interest. In the context of AUTCOMS, they will have a focus on automation and its real-time specifics.

Acknowledgements

The author would like to thank Edin Arnautovic, Dietmar Dietrich and the anonymous reviewers for useful comments on earlier drafts of this article.

References

Becker, L. B., Pereira, C. E. (2002): SIMOO-RT – An object-oriented framework for the development of real-time industrial automation systems. IEEE Trans. on Robotics and Automation 18(4): 421–430.

Davis, A. M. (1993): Software requirements: objects, functions, and states. Englewood Cliffs, NJ: Prentice Hall.

Ebner, G., Kaindl, H. (2002): Tracing all around in reengineering. IEEE Software, May/June 2002: 70–77.

INCOSE (2000): Systems Engineering Handbook, Vers. 2.0. International Council on Systems Engineering, Seattle, WA, July 2000.

Jacobson, I., Booch, G., Rumbaugh, J. (1999): The unified software development process. Reading, MA: Addison-Wesley.

Kaffenberger, R. (2003): Will object-oriented systems-engineering reshape requirements engineering? In: Proc. of the 13th Annual Int. Symp. of the Int. Council on Systems Engineering, Arlington, VA, June/July 2003 INCOSE: 583–595.

Kaindl, H. (1997): A practical approach to combining requirements definition and object-oriented analysis. Annals of Software Engineering 3: 319–343.

Kaindl, H. (1999): Difficulties in the transition from O-O analysis to design. IEEE Software: 94–102.

Kaindl, H. (2001a): A design process based on a model combining scenarios with goals and functions. IEEE Trans. on Systems, Man, and Cybernetics (SMC), Part A 30(5): 537–551.

Kaindl, H. (2001b): Using hypermedia in requirements engineering practice. The New Review of Hypermedia and Multimedia 7: 185–205.

Kaindl, H., Brinkkemper, S., Bubenko Jr., J. A., Farbey, B., Greenspan, S. J., Heitmeyer, C. L., Leite, J. C. S. do P., Mead, N. R., Mylopoulos, J., Siddiqi, J. (2002): Requirements engineering and technology transfer: obstacles, incentives and improvement agenda. Requirements Eng. 7(3): 113–123.

Kaindl, H., Carroll, J. M. (1999): Introduction: symbolic modeling in practice. Communications of the ACM. 42(1): 28–30.

Kaindl, H., Kramer, S., Kacsich, R. (1998): A case study of decomposing functional requirements. In: Proc. 3rd Int. Conf. on Requirements Engineering (ICRE '98), Colorado Springs, CO, April 1998. IEEE: 156–163.

Loucopoulos, P., Karakostas, V. (1995): System Requirements Engineering: London: McGraw-Hill.

Mannion, M., Keepence, B., Kaindl, H., Wheadon, J. (1999): Reusing single system requirements from application family requirements: In: Proc. 21st Int. Conf. on Software Engineering (ICSE-99), Los Angeles, CA, May 1999. ACM: 453–462.

Mannion, M., Lewis, O., Kaindl, H., Montroni, G., Wheadon, J. (2002): Representing requirements on generic software in an application family model. In: Proc. 6th Int. Conf. on Software Reuse (ICSR-6), Vienna, June 2000. Berlin: Springer: 153–169.

Nuseibeh, B., Easterbrook, S. (2000): Requirements engineering: a roadmap. In: Future of Software Engineering, special volume of Proc. 22nd Int. Conf. on Software Engineering (ICSE-2000), Limerick, Ireland, June 2000. ACM.

Oliver, D. W. (1997): Engineering of complex systems with models. IEEE Trans. on Aerospace and Electronic Systems 33(2): 667–685. ■