A Comparative Case Study with Industrial Requirements Engineering Methods

Antje von Knethen[1], Erik Kamsties[2], Ralf Reussner[3], Christian Bunse[2], Bin Shen[1]

[1]Department of Computer Science, University of Kaiserslautern,
D-67653 Kaiserslautern, Germany
vknethen@informatik.uni-kl.de
[2]Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
kamsties@iese.fhg.de
[3]Department of Computer Science, University of Karlsruhe
Am Fasanengarten 5, D-76128 Karlsruhe, Germany
reussner@ira.uka.de

Abstract

Numerous requirements engineering methods have been proposed to improve the quality of requirements documents as well as the developed software and to increase customer satisfaction with the final product. In this paper, we report on an explorative case study in the area of reactive systems with eight requirements engineering methods from a wide spectrum, namely, BSM, OCTOPUS, ROOM, SA/RT, SCR, SDL, UML (with OMT process), and Z. Our major finding is that the structuring mechanisms provided by a requirements engineering method, like hierarchies (e.g., ROOM) or views (e.g., UML), are directly related (1) to the number of problems found in the informal requirements during the creation of a requirements specification as well as (2) to the understandability of the final requirements specification.

1 Introduction

Case studies with requirements engineering methods have a long tradition. Comparative studies of different methods have been performed, for instance, at the International Workshop on Software Specification and Design (IWSSD) since the mid-80s. We use the term *requirements engineering method* (RE method) to describe approaches that define the type of knowledge to be captured in a requirements specification, and provide a more or less detailed description of how to elicit, document, and check the requirements. The focus of our case study was on *industrial* RE methods for reactive systems, meaning RE methods developed and used in an industrial environment, or developed at academia and applied successfully in industrial case studies.

Reactive systems become increasingly wide-spread in telecommunications, automotive systems, avionics, customer products (e.g., TV remote control), and several other areas, and many RE methods have been proposed for their development. However, the question is which one to choose for a particular project.

The goal of our case study was to assess three important aspects of RE methods, namely (1) their applicability with respect to a particular environment, (2) their impact on the specification quality, and (3) their effectiveness. In the case study, we applied eight industrial RE methods from a wide spectrum, namely BSM (part of Cleanroom), OCTOPUS, ROOM,

SA/RT, SCR, SDL, UML, and Z. The case study had an explorative nature, thus we started with the vague hypothesis that the RE methods differ in their applicability, impact on specification quality, and effectiveness. Our aim was to identify the key characteristics of RE methods that affect these aspects (e.g., has the degree of formality provided by an RE method an impact on their applicability?).

We conducted the study with graduate students in a university environment. We developed on our own specification exemplar (i.e., problem) based on the Tamagotchi toy [Ban97]. The reason for developing our own exemplar was two-fold: First, the Tamagotchi is a typical example of a consumer product, thus, our results can be generalized to similar products. Second, most students probably know popular exemplars, like the cruise control, or lift exemplars, from many text books (thus, cheating would be possible).

In work related to ours, Ardis et.al. [ACJ+96] developed a set of criteria and their own exemplar particular to their system and software development environment (AT&T's 5ESS switch software). Lewerenz and Lindner [LL94] collected several specifications of the Production Cell exemplar. In a similar way, Abrial, Boerger, and Langmaack [ABL96] edited a set of specifications of the Steam Boiler Control. Our work differs from the above in that we assess not only the resulting specification, but also the associated activities (e.g., creation, checking). An RE method helps not only to state the requirements precisely, it also allows to find problems in informal requirements while creating the specification. The second feature was only studied by Wing [Win88] post-mortem in the resulting specifications of the fourth IWSSD workshop (1986). We studied this phenomena in-situ (i.e., during the creation of specifications).

The remainder of this paper is organized as follows. In section 2, we describe the design of our comparative case study. Section 3 contains the results of the case study. We finish in section 4 with a summary of our results and some conclusions.

2 Case Study Design

This section describes the research questions underlying the design of our case study, the RE methods that were used, the Tamagotchi exemplar, and the case study procedure.

2.1 Research Questions

The long term goal of our research is to evaluate industrial RE methods to identify areas for evolutionary improvement, instead of developing a complete new approach at academia and trying to transfer it to industry (revolutionary improvement). Past experience indicates that the revolutionary approach for transfer does not work well. The basic questions that should be answered by our case study are the following:

1. *How well is a RE method applicable in a particular environment?*

   We examined three criteria: learning effort, expressive power of notation, and process support with respect to our university environment. In different environments, other criteria might be useful. Ardis et.al. [ACJ+96], for instance, examined the question whether an RE method is compatible with AT&T's existing specifications.

2. *How does an RE method impact the quality of the requirements specification?*

   We focussed on the "-abilities" of the resulting requirements specifications that are likely to be different: understandability, traceability, modifiability, testability, and

implementability[1]. Our aim was to understand how particular properties of the RE methods (e.g., behavioral model is partitioned according to classes vs. unrestricted partition) impact the "-abilities" of the specifications created.

3. *How effective and efficient are activities around the requirements specification?*

   Activities might be explicitly supported by the RE method or just implicitly prescribed by the notation. We basically looked at the creation of the requirements specification, and recorded the number of problems that were spotted in the informal requirements document, the creation effort, and the size of the final specification (i.e., the number of pages and diagrams). To better understand the effort and size, we also recorded the number of views (or models, respectively).

Questions 1 and 2 can be answered qualitatively. Question 3 can be answered quantitatively with effort data and numbers of problems found during the creation.

2.2 Requirements Engineering Methods

In this section, we describe the RE methods used in the case study. All RE methods claim to be qualified for the development of reactive systems with soft real-time requirements and have been applied in industry. The number of RE methods was limited by the number of students that took part in the case study. To cover a wide spectrum of the different kinds of methods, we selected methods of two different classes: function-oriented (BSM, SCR, SA/RT, and Z), and object-oriented methods (OCTOPUS, ROOM, SDL, and UML). Furthermore, these RE methods differ in their degree of formality: UML, OCTOPUS, BSM, ROOM, SDL, and SA/RT are semi-formal methods, SCR, and Z are formal methods. Please note that not all of these are really *methods* (i.e., provide also process support); SCR, SDL, UML, and Z are pure notations. In the case of SCR and Z, the application is more or less obvious, in the case of UML, we adapted the OMT process, and in the case of SDL we partially used [BH93].

BSM (Box Structure Method) was developed by Mills at IBM as part of the Cleanroom Approach in the 80s [Mil88]. It is a specification and design method based on functional decomposition. We used only the so-called "black box", but not the state or clear box. *SCR* (Software Cost Reduction) has been developed by the NRL (Naval Research Laboratory) since 1980 [Hei96]. It is a specification technique for embedded systems and does not give any methodological support. *SA/RT* (Structured Analysis/Real-Time) is a functional method supporting the specification, design, and implementation of real-time systems. It was developed by Hatley and Pirbhai in 1987 [HP87]. *Z* [PS96] is a general-purpose, formal, model-based specification technique. It was developed by the University of Oxford in the late 80s. There is no prescriptive process description, but a number of text books offer general methodological support.

OCTOPUS [AKZ96] was developed by Nokia in 1996. It is an object-oriented method that integrates and adapts OMT (Object Modeling Technique) [RBP+91] and Fusion [Col94] for analyzing, designing, and implementing real-time systems. Likewise, *ROOM* (Real-time Object-Oriented Modeling) is an object-oriented method for the development of real-time systems [SGW94]. The method was developed by Bell Northern Research Ltd. in 1994. *SDL*

[1] We assumed that correctness, completeness, and consistency depend heavily on the skills of the requirements engineer, not so much on the RE method. Therefore, we did not consider these qualities in our case study.

(Specification and Description Language) was developed by the organization preceding the International Telecommunication Union (CCITT) in the late 60s and has been improved until now. Since version SDL 92, the language is object-oriented. Its main application is the specification, design, and implementation of telecommunication systems. To simplify the specification, the employment of Message Sequence Charts is recommended. SDL does not include process guidance, but processes for SDL are available (e.g., [BH93]). *UML* (Unified Modeling Language) was developed by Booch, Rumbaugh and Jacobson at Rational Software Cooperation in 1997 [BRJ97]. The language is based on OMT, the Booch method [Boo91], and OOSE [Jac94]. The goal of the development was to standardize the different object-oriented methods for specification, design, and implementation of software systems. UML does not include methodological support, but processes are available (e.g., OMT). The language allows the modeling of all kinds of systems from their specification to their implementation.

All methods are supported by several tools. In our case study we did not employ these tools due to their high learning effort.

2.3 Subjects and Materials

The case study was performed in a seminar at the University of Kaiserslautern in the winter term 97/98. Eight graduate students took part. Each student chose one method to examine. Most of the students had only minor experience in developing requirements specifications and some experience in reading specifications. The seminar was supervised by five experts who are familiar with the different RE methods used.

At the beginning of the case study, the students received an informal requirements document for the Tamagotchi toy. The document was three pages long and contained 36 requirements in natural language (German). Each requirement consisted of no more than two sentences. The requirements document contained a number of known problems (i.e., inaccuracies, inconsistencies, incompleteness).

The Tamagotchi is a reactive system simulating the life cycle of a small chicken. The chicken has a number of development stages (e.g., Egg, Babychi, and Kutchitamatchi). The user has to care for the chicken, for example, by playing with it or feeding it. The life time and the development stages depend on the functionality the user executes and the internal state of the chicken.

The Tamagotchi system consists of a display, a buzzer, three buttons and the software system itself. Data, like the level of happiness or the development stage of the chicken, are output on the display. With the buttons, the user is able to execute some functionality, like feeding the chicken or playing with it. The Tamagotchi can make itself felt with the buzzer by giving a signal tone.

The main challenges of the Tamagotchi specification task were: First, the modeling of the user interface functionality. The informal requirements were written in terms of the user interface (e.g., “the Tamagotchi shall be fed by selecting the menu item ‘feed’ with the left button and confirming with the middle button...”). Second, the modeling of the timing requirements. The timing requirements of the Tamagotchi are of two types: periodic actions (e.g., “the Tamagotchi shall go to bed each day at 8 pm”), and durations/timeouts (e.g., “the menu shall be closed after five seconds if no key was pressed”). Third, the natural structure of the problem domain. The Tamagotchi “problem domain” exposes a natural separation of different behavioral issues (e.g., feeding, playing, etc.), different structural items (display,

buttons, etc.), and different operational modes (Babychi, Kutchitamachi, etc.), which can be used to structure a specification.

2.4 Procedure

The Tamagotchi case study was performed in three phases. The task of the first phase was to become familiar with the RE method the student had chosen. Therefore, s/he had to read literature, and write a short essay on the RE method (e.g., background, intention and constructs of the method).

In the second phase, the students developed specifications based on the given informal requirements document. Problems detected within this document were recorded and resolved individually. Furthermore, the students developed traceability matrices showing the relations between the informal requirements and the specification.

In the third phase, the students examined the maintainability of their specifications. Therefore, they got a change request for the informal requirements and they had to describe the impact on their specification. In this phase, the specifications were also cross-reviewed among the students.

3 Results

The results presented are derived from the data collection forms, from interviews with the students, and from the students' concluding presentations. The following discussion of results is organized along the lines of our three research questions given in section 2.1.

3.1 Applicability (Question 1)

We assessed three issues: learning effort, expressive power of notation, and process support.

Learning effort

The learning effort (i.e., reading and understanding the literature) differed between 20 and 32 hours and was correlated basically to the richness of the notation (i.e., the number of views and/or the size of the offered notation). BSM and SCR were claimed as quick and easy to learn, due to their simple notations. The learning effort for the other RE methods (SDL, UML, ROOM, Z, SA/RT, and OCTOPUS) was acceptable with respect to a university seminar.[2]

Expressive power of notation

The expressive power of a notation was judged with respect to the challenges of the Tamagotchi specification task, which are: user interface, timing requirements, and natural structure of the problem domain (see section 2.3):

- *User interface*

  None of the RE methods provide any hints or heuristics to map the phenomena of user interfaces[3] (and real-world phenomena in general) to the constructs of the notation. We found this mapping critical for two reasons. First, the informal requirements were written in terms of the user interface, therefore, the specification should reflect this to be

[2] We do not provide detailed effort data here, because there were no significant differences (i.e., a "ceiling effect" occurred), since the students had a limited amount of time available for learning the RE method (3 weeks).

[3] The term "user interface" comprises *all interfaces to the user* of the Tamagotchi, which are the buttons, the display, and the buzzer.

understandable and complete. Second, the mapping (in other words, choosing the right abstraction level) has a significant impact on the size of the resulting specification. BSM appears to be most sensitive to the issue of the abstraction level; specifying the black box can become infeasible, if the number of stimuli is too high. The reason is that BSM do not provide notational means to group stimuli, which are essential to deal with a high number of stimuli. Choosing the abstraction level was straight forward for SCR, however, the description of the Tamagotchi's multi-functional display was clumsy, because SCR's environmental variables are too restrictive to describe the variability of the display. The abstraction level was not a problem with OCTOPUS either. But contrary to SCR, OCTOPUS offers more flexibility for describing outputs (e.g., distinction of actions and activities), which helps describing the user interface better. SDL, ROOM, and UML are similar to OCTOPUS. SA/RT was the only RE method that allows a direct and explicit specification of the buzzer's signal tones by timing diagrams.

- *Timing requirements*
  The timing requirements of the Tamagotchi are of two types: periodic actions and durations/timeouts. We did not find simultaneous support for both types of timing requirements, except in SA/RT. In SA/RT, timing diagrams model durations, periodic actions are described informally in the process activation table. For other methods, typically, a 'clock' (for periodic actions) and 'timer' (for durations/timeouts) must be invented in the specification. A slight exception was SCR, which provides a construct for durations, but nothing for periodic actions. In ROOM and SDL, a timer is predefined.

- *Natural structure of the problem domain*
  The natural structure of the Tamagotchi problem domain could not be reflected with the part of BSM that we applied (black box). It remains an open question whether the procedural elements (i.e., sequence, alternative, iteration, concurrency) offered by the BSM clear box would help structuring the problem domain. SCR offered so-called "mode classes" to structure the behavior according to operation modes. However, this was not considered sufficient, because a large number of modes (26) were identified (i.e., the major problem domain structure was lost). OCTOPUS offered two concepts: first, "subsystems" allow for a coarse-grained partition of the system, second, "statecharts" capture one coherent behavioral aspect of a subsystem (independent of the subsystem's object model). UML, ROOM, and SDL allow structuring the system behavior by classes, actors, or blocks, respectively. During the creation of the specification with OCTOPUS, UML, ROOM, and SDL, the main challenge was the determination of a "good" partition of the system into subsystems/classes/actors/blocks. SA/RT offers abstraction through hierarchical decomposition of its flow diagrams. In Z, schemata serve as a kind of macro, which can be used to structure the specification.

Process support

The process support was relatively sparse for all RE methods. Most methods basically describe in which order the different descriptions, models, or views shall be produced. The question of how to map real world phenomena onto the notational elements was often tackled superficially. The problem of how to identify "good" objects is well known, but identifying "good" states turned out to be a problem, too. SA/RT, ROOM, and SDL offer hierarchical structuring mechanisms (i.e., structuring by refinement). It is unclear when to finish the refinement (i.e., specification) process. Therefore, some of the developed specifications are

more detailed than others. Similarly, the support for checking the specifications is sparse. BSM, ROOM, SDL, UML and SCR provide some mechanisms (based on their requirements model) to check specifications for internal completeness and consistency. For SA/RT and OCTOPUS, there are guidelines for internal consistency checking, which proved to be useful during application.

3.2 Impact on Specification Quality (Question 2)

The desired qualities of a requirements document are commonly accepted. However, there is no consensus on how to measure those qualities. Therefore, our discussion is based on purely subjective observations, and in the following we focus on understandability and traceability to keep the discussion short. One major finding of this case study is that the structuring mechanisms offered by an RE method affect the understandability of the final specification. The traceability of a specification depends on the paradigm of the applied RE method. Since the informal requirements document were written in a functional style, traceability could be documented in the case of a function-oriented black box specification (i.e., BSM) easier than with any other RE method.

Understandability

Each student found his/her own specification easier to understand than the specification s/he had to review. The rationales behind this were interesting: The authors of the "flat" specifications (e.g., BSM and SCR) argued that it is easier to tell what the response of the system is to a particular stimulus compared to hierarchical specifications (e.g., SA/RT and SDL). Vice versa, the authors of hierarchical specifications argued that it is easier to locate a particular topic in the hierarchical specification, because one can follow the hierarchical decomposition of the specification, instead of reading sequentially through the specification. It seems to us that both specification styles facilitate understanding, but unfortunately, they are mutually exclusive. Similar to hierarchical specifications, the stimulus-response relationship is not obvious in object-oriented specifications, because the system behavior is distributed over several interacting objects.

Traceability

The task of all students was to prepare a traceability matrix, which describes the relation between informal requirements and the specification. Remarkable was the BSM specification, because each single reaction upon a stimulus can be easily referenced. In contrast, graphical RE methods only allow to reference full diagrams without sacrificing the readability of the diagrams. We found a *1:1* traceability relationship between informal requirements and BSM specification, as opposed to a *n:1* or *n:m* traceability relationship with the other RE methods.

3.3 Effectiveness and Efficiency (Question 3)

We measured the specification effort, size, and number of problems found in the informal requirement quantitatively. Table 1 "Quantitative Results" summarizes the results. The second major finding of this case study is that the structuring mechanisms offered by an RE seem to be the major factors that influence effort, size, and number of problems spotted in the informal requirements document during the specification process.

RE method	BSM	SCR	Z	SA/RT	SDL (+MSC)	ROOM	UML	OCTOPUS
Structure of specification	flat	flat	hierarch.	hierarch.	hierarch.	hierarch.	hierarch. (state diagram)	hierarch. (statecharts)
#Views/Diagram Types	1 / -	1 / -	1 / -	3 / 5	2 / 3	2 / 4	3 / 5	3 / 4
# of Abstraction Levels	1	1	max. 3	3	max. 5	max. 6	max. 3	max. 5
Effort (h)	33	50	73	52	90	85	90	77
Size (pages/ # diagrams)	15 / -	20 / -	17 / 24	27 / 26	18 / 43	52 / 36	45 / 39	37 / 14
# of problems	15	15	5	24	37	35	35	20

Table 1: Quantitative Results

When we examined the characteristics of the RE methods in a more detailed manner, we found that hierarchies and the number of views (or models), of course, increase the effort and size of the specification. Quite interesting was the fact that these factors also influence the number of problems found with an RE method. A reason might be that flexible refinements of particular issues and the analysis from different view points lead to a more thorough analysis of informal requirements.

Another interesting trend is that some object-oriented methods (SDL, UML, and ROOM) seem to lead to the detection of more problems than OCTOPUS or SA/RT. The reason for this might be that partitioning of the system into objects (respectively blocks for SDL) lead to a more thorough analysis of behavior than modeling the system as a whole (the behavioral model in OCTOPUS is partitioned according to large subsystems, but not to objects). The interaction between processes in SA/RT cannot be analyzed as detailed as object interactions with the help of sequence diagrams.

Formal methods (SCR, Z) did not perform better than semi-formal methods in spotting defects in the informal requirements document. We consider the quantitative results for Z as outliers, since we could not find any reason why the application of Z leads to the detection of less problems than the application of any of RE method. Therefore, we conclude that the learning period (3 weeks) was probably to short for students, which had not previous exposure to formal methods.

The data provided in Table 1 must be used carefully. It is difficult to determine why a participant found a particular number of problems (e.g., ‘37’) in the informal requirements. One reason could be the support of the method used to find problems, but it is more reasonable to assume variances in the performance of individuals. The same is true for the effort needed to create the specification. Therefore, the data cannot be used for assessing a single RE method, but only for identifying trends in the data as we did above.

4 Summary and Conclusion

We compared eight industrial RE methods, namely, BSM, OCTOPUS, ROOM, SA/RT, SCR, SDL, UML (with OMT process), and Z, in an explorative case study with graduate students in a university environment. Each student had to learn a RE method. Then s/he had to create a specification from given requirements in natural language.

The goal of our case study was to assess three important aspects of industrial RE methods for reactive systems, namely their *applicability* with respect to a particular environment, their *impact on the specification quality*, and their *effectiveness*.

Applicability. Among the RE methods examined, BSM and SCR require a relatively low learning effort compared to the others. The basic concepts behind BSM, SCR, and Z make it more difficult to specify the user interface than the concepts behind the other RE methods. Timing requirements were typically operationalized by a 'clock' (for periodic actions) and by a 'timer' (for durations/timeouts). A notable exception was SA/RT: its timing diagrams model durations, periodic actions are described informally in the process activation table. The process support was relatively sparse for all RE methods.

Impact on specification quality. The type of a specification (flat vs. hierarchical) influences the understandability of a specification. A hierarchical or object-oriented specification (i.e., SA/RT, SDL, ROOM, UML, OCTOPUS) makes it easier to locate particular informal requirements in the specification, which is one aspect of understandability, because the hierarchy guides the reader quickly through the specification. A flat specification (i.e., BSM, SCR, Z) forces the reader to read the specification sequentially, which requires more time. However, a flat (black-box) specification makes it easier to grasp the system's input-output behavior, which is another aspect of understandability, because inputs are directly linked to outputs. Tracing an input across several interacting objects (or through a hierarchy) to the according outputs is not that easy.

Effectiveness. Our quantitative data shows that students applying RE methods with views or hierarchies (i.e., SA/RT, SDL, ROOM, UML, OCTOPUS) needed more time, but they found significantly more problems in the informal requirements than students producing a flat specification (i.e., BSM, SCR, Z). Most effective in finding problems were those students, which applied RE methods that force to partition the system into objects/blocks and to analyze the interactions between these objects/blocks (i.e., SDL, UML, and ROOM).

Our results show that each method has its own strengths. Therefore, we do not designate a winning method. Hence, the interesting question is, how to combine the strength of several RE methods (e.g., the advantages of a flat and a hierarchical specification).

We believe that the results of our case study can be transferred to an industrial environment to some extent, for two reasons. First, the Tamagotchi represents a wide area of consumer products, such as remote controls or mobile phones. Second, industry is hiring much people these days, thus, it is not unusual that a young professional has to develop a specification without much experience in applying the particular method. We believe that the behavior of these young professionals are comparable to some extent to the behavior of the graduate students participating in our case study. However, our results are less expressive for real professionals.

With respect to research, we believe that the results of our explorative case study provide interesting hypotheses for more focused and in-depth studies.

Acknowledgments

We thank the students participating in the seminar: Jens Bonnermann, Holger Damczyk, Arne Koennecker, Stephan Kupjuweit, Mirjam Krementz, Juergen Petry, Sascha Schwarz, and Andreas Werner. Furthermore, we thank our colleagues in the Software Engineering Research Group at the University of Kaiserslautern and in the Fraunhofer Institute for their contributions, especially Prof. H. Dieter Rombach and Dr. Barbara Paech for their advice.

Literature

[ABL96] J.-R. Abrial, E. Boerger, and H. Langmaack (Eds.). *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*. Lecture Notes in Computer Science 1165. Springer, 1996.

[ACJ+96] Mark A. Ardis, John A. Chaves, Lalita Jategaonkar Jagadeesan, Peter Mataga, Carlos Puchol, Mark G. Staskauskas, and James Von Olnhausen. A framework for evaluating specification methods for reactive systems : Experience report. *IEEE Transactions on Software Engineering*, 22(6):378-389, June 1996.

[AKZ96] Maher Awad, Juha Kuusela, and Jurgen Ziegler. *Object-Oriented Technology for Real-Time Systems: A Practical Approach Using OMT and Fusion*. Prentice Hall, 1996.

[Ban97] Bandai. *Das Original Tamagotchi Buch*. TAMAGOTCHI & BANDAI, 1997.

[BH93] R. Braek and O. Haugen. *Engineering Real-time Systems: An object-oriented Methodology using SDL*. Prentice Hall, London, 1993.

[Boo91] Grady Booch. *Object Oriented Design with Applications*. Benjamin/Cummings, New York, 1991.

[BRJ97] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language, Version 1.0*. Rational Software Corporation, 1997.

[Col94] Derek Coleman. *Object oriented development: the Fusion method*. Prentice Hall, London, 1994.

[DKZ97] Thomas Deiss, Martin Kronenburg, and Dirk Zeckzer. A catalogue of criteria for evaluating formal methods and its application. SFB report 4/97, University of Kaiserslautern, Kaiserslautern, Germany, 1997.

[Hei96] Constance L. Heitmeyer, Ralph D. Jeffords and Bruce G. Labaw. Automated Consistency Checking of Requirements Specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231-261, July 1996.

[Hen80] Kathryn L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Transactions on Software Engineering*, SE-6(1):2-13, January 1980.

[HP87] Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House Publishing, 1987.

[Jac94] Ivar Jacobson. *Object-oriented software engineering: a use case driven approach*. Addison Wesley, New York, 1994.

[LL94] Claus Lewerenz and Thomas Lindner (Eds.). Case study "production cell". FZI-Publication 1/94, Forschungszentrum Informatik (FZI), Universitaet Karlsruhe, Germany, 1994.

[Mil88] Harlan D. Mills. Stepwise refinement and verification in box-structured systems. *IEEE Computer*, June 1988.

[PS96] Ben Potter and Jane Sinclair. *Formal Specification and Z*. Prentice Hall, London, 1996.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[SGW94] B. Selic, G. Gullekson, and P.T. Ward. *Real-Time Object-Oriented Modeling*. John Wiley & Sons, New York, 1994.

[Win88] Jeannette M. Wing. A study of 12 specifications of the library problem. *IEEE Software*, pages 66-76, July 1988.