Dagstuhl-Seminar 99241

Requirements Capture, Documentation, and Validation

June, 13-18 1999

E. Börger, B. Hörger, D.L. Parnas, H.D. Rombach

Table of Contents

Summary

The goal of the workshop, namely to bring together software engineering researchers from academia and software engineers from industry to compare the state of industrial practice and academic research for capturing, documenting and validating software requirements, has been reached.

After two days of short introductory presentations (see the abstracts below), with ample time for critical discussion, we had two days of intensive discussion in working groups.

The three themes

- Integrating Process, Tools and Formal Methods (moderator Connie Heitmeyer),
- Requirement Engineering Process, Evolution of Requirements and Traceability (moderator Barbara Paech),
- The Light Control Case Study (moderator E. Börger)

were selected by the participants on Tuesday evening, the results obtained were presented to all participants during the closing session on Friday morning. Reports by the moderators of the working groups can be found below.

The focus of the presentations and discussion was on the industrial strength of the used methods and on their relevance for the production of large software.

To make sure that the discussion was suitably concrete, the workshop made extensive use of a case study that could be discussed in detail. The example, taken from the area of building automation, was a light control system. A more detailed discussion of this system can be found below.

We thank Erik Kamsties, Antje von Knethen, and Barbara Paech for their help in preparing the seminar and for maintaining the web page for the case study. Thanks also to the Dagstuhl team for creating the pleasant atmosphere for our work.

Egon Börger (Universita di Pisa)
David Parnas (McMaster University, CDN)
Bärbel Hörger (DaimlerChrysler, Ulm)
Dieter Rombach (Universitaet Kaiserslautern)

Abstracts of Talks in Order of Appearance

Using SCR Method to Capture, Document, and Verify Computer System Requirements[1]

Constance Heitmeyer Naval Research Laboratory (Code 5546), Washington, DC 20375 USA, heitmeyer@itd.nrl.navy.mil

Since 1993, our group at NRL has been developing a set of software tools [5, 3, 1] for specifying and analyzing computer system requirements using the SCR (Software Cost Reduction) method, a tabular method introduced in the A-7 project [6, 7]. The tools include a specification editor for creating a requirements specification; an automated consistency checker to detect missing cases, unwanted nondeterminism, and other application-independent errors [5]; a simulator to symbolically execute the specification to ensure that it captures the users' intent [4]; and a model checker to detect violations of critical application properties [1, 3]. Recently, groups at NASA and Rockwell Aviation as well as our group at NRL have used the SCR method to detect serious errors in requirements specifications of practical systems [2, 9, 3, 8].

This talk discusses the SCR approach to specifying requirements within the context of the Parnas-Madey Four Variable Model (FVM) [10]. First, a description is given of how our method and tools can be used to specify NAT and REQ, two relations defined on the environmental quantities - i.e., the monitored and controlled quantities- of the FVM. The relation NAT describes the constraints imposed on the monitored and controlled quantities by physical laws and the system environment. The relation REQ describes the additional constraints on the controlled quantities that the system must enforce.

Next, an approach is sketched for capturing the required behavior after the system designers have selected the system's I/O devices. The system will use these devices to sample the values of the monitored quantities and to set the values of the controlled quantities. We propose to divide the software into three modules: the input module, the output module, and the device-independent module. The input module uses the values read from input devices to compute estimates of the monitored quantities, while the output module writes the values computed by the system for the controlled variables to the output devices. Both the input module and the output module are examples of device-dependent modules. The third module, the device-independent module, uses the estimates of the monitored quantities produced by the input module to compute the values of the controlled quantities. The required behavior of the device-independent module is described by the relation REQ. The SCR method and tools can be used to specify, verify, and validate the input and output modules in the same manner as they are currently used to specify, verify, and validate the two relations, NAT and REQ.

[1] This research is supported by the Office of Naval Research.

Finally, examples are presented which illustrate the SCR approach to requirements specification. These examples are taken from the informal description of the light control system used in the case study. To illustrate how the SCR method can be used to specify the input module, we show how sensor data can be used to estimate the value of a given monitored quantity, the quantity which indicates whether a given room is occupied.

Our approach is motivated by a new project in which we will be using the SCR requirements method and tools to help build a helicopter trainer for the U.S. Navy. To begin, we will develop an SCR-style requirements specification of the trainer using our current method and tools to specify the monitored and controlled quantities and the relations NAT and REQ. An important goal of the project is to do hardware-in-the-loop simulation. To achieve hardware-in-the-loop simulation, our plans are to extend the existing toolset, including the current simulator. Currently, our simulator uses the specification to compute the values that are assigned to controlled quantities as the monitored quantities change. Thus, it implements the device-independent module described above. We plan to extend the simulator to read actual sensor data and to write the computed values of the controlled quantities to actual actuators. To do so, the current toolset will be extended to support the specification of the input and output modules described above.

References

1. R. Bharadwaj and C. Heitmeyer. Model checking complete requirements specifications using abstraction. Automated Software Eng. J., 6(1), January 1999.
2. S. Easterbrook and J. Callahan. Formal methods for verification and validation of partial specifications: A case study. Journal of Systems and Software, 1997.
3. C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. IEEE Trans. on Softw. Eng., 24(11), November 1998.
4. C. Heitmeyer, J. Kirby, Jr., and B. Labaw. Tools for formal specification, verification, and validation of requirements. In Proc. 12th Annual Conf. on Computer Assurance (COMPASS '97), Gaithersburg, MD, June 1997.
5. C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. ACM Trans. on Software Eng. and Methodology, 5(3):231-261, April-June 1996.
6. K. Heninger, D. Parnas, J. Shore, and J. Kallander. Software requirements for the A-7E aircraft. Technical Report 3876, Naval Research Lab., Wash., DC, 1978.
7. Kathryn L. Heninger. Specifying software requirements for complex systems: New techniques and their application. IEEE Trans. Softw. Eng., SE-6(1):2-13, January 1980.
8. J. Kirby, Jr., M. Archer, and C. Heitmeyer. Applying formal methods to an information security device: A case study. In Proc., Symp. on Protecting NATO Information Systems in the 21st Century, Wash. DC, October 1999.
9. S. P. Miller. Specifying the mode logic of a flight guidance system in CoRE and SCR. In Proc. 2nd ACM Workshop on Formal Methods in Software Practice (FMSP'98), 1998.
10. David L. Parnas and Jan Madey. Functional documentation for computer systems. Science of Computer Programming, 25(1):41-61, October 1995.

Practical Formalisms

Jo Atlee, Department of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada

Several studies have shown that requirements are often unknown, misunderstood, or miscommunicated. Our objective is a specification notation that

- has a precise semantics (preferably one that allows automated checking and analysis),
- can be read and reviewed by both domain experts and software professionals, and
- encourages the requirements writer to consider completeness (i.e., to consider how the system should respond to every input)

"Practical Formalisms" (a term coined by David Harel) come close to meeting these objectives. Practical formalisms are specification notations that have a formal, mathematical model; this ensures that a specification written in the notation has a single interpretation. They encourage the use of abstraction and separation of concerns to simplify and decompose a specification into a set of functions and relations, each of which is smaller, simpler, and easier to consider than the original problem. Also, they have diagrammatic constructs (e.g., tables and/or graphs) for expressing the individual functions and relations in an easy-to-read and checkable format. Example practical formalisms include SCR, CoRE, Statecharts, and RSML.

These notations and their semantics are still evolving. In fact, when we specified in SCR a portion of the Dagstuhl Light Control System, we recognized the need for new notation-supported abstractions:

- Parameterization - so that the behaviour for rooms of the same type can be specified once
- Inheritance - so that the behaviour common to all room types could be specified once, and behaviours that vary according to room type could be specified as extensions to the common behaviour.

An Industrial Experience: Software Quality Models for Quality Requirements at DaimlerChrysler AG

Kurt Schneider, Stefanie Lindstaedt, Thomas Beil.
DaimlerChrysler Research Center Ulm

Quality requirements are widely neglected in industrial software projects – and they are often ignored by research. As a consequence, quality requirements do not get as much attention in contracts and in acceptance procedures. But since software quality is at stake, we need a systematic way to treat, trace, and advocat quality requirements. With our quality model approach, we break down abstract quality goals (e.g., "efficiency, reliability, usability") to questions and finally, to testable criteria. This approach serves several requirements demands. We used it to allow managers, developers, and testers to negotiate and gain a common understanding. However, each

quality goal calls for large amounts of specific knowledge. We tackle with this growing demand mainly by an approach of systematically learning from experiences.

Using Formal Methods in UML to Verify Requirement Properties on Specifications

Hugues Martin, GEMPLUS, France
hugues.martin@gemplus.com

Requirements need to be understandable by all the stakeholders of the software development process. At the same time, they need to satisfy properties such as consistency and unambiguity. UML, using graphical representation and natural language, provides a clear notation usable by all the stakeholders to write understandable specifications. However, UML is a semi-formal language which does not support automatic verification techniques on the models. One possibility would be to integrate formal methods such as B or SDL in UML. Then it would become possible to apply verification techniques on some parts of the model, and to preserve the readability of the specifications. This integration needs to take into account the maintainability of the UML model and the formal model, and to facilitate the reusability of components by clearly identify interfaces between the different representations.

Lessons Learnt in Transferring Formal Requirements Validation Techniques to Industry

Steve Easterbrook, University of Toronto
sme@cs.toronto.edu

This talk summarizes experience from a number of formal methods case studies over the last four years at NASA. The case studies all concern the independent verification and validation for safety critical systems. In each case, formal methods were applied in a lightweight way to improve clarity, automate consistency checking, to animate the requirements and validate the behavior specified. Methods used include Spin, SCR and PVS. The main lessons are that these methods are effective at finding important errors that are not typically found using existing inspection-based approaches. However, the existing methods are weak at analyzing change and dealing with refinements of specifications. The talk will also discuss a number of technology transfer issues such as what types of expertise are needed to apply these methods, where to apply them, and how to measure the benefits.

Application of Ignorance to Find Errors in the Case Study Requirements Specification

Daniel M. Berry, Technion and University of Waterloo
dberry@csg.uwaterloo.ca

***Comment**: The following text is not an abstract but an email that was sent to all participants of the Dagstuhl seminar.*

My planned contribution to the Dagstuhl Seminar was to apply ignorance (as per my ‘‘Importance of Ignorance in Requirements Engineering”) to help identify problems in the informal specification that must be resolved before a complete and consistent specification can be written. That is, I read the informal specification of the light control system as someone who knows very little about such systems and had lots and lots of questions borne of this ignorance. I attached these questions as notes to the PDF document containing the informal specification.

My original plan was and still is to present these questions at the seminar to see if they cause anyone to notice problems in his or her own contribution to the seminar. Now that I have completed my contribution early, I thought that it might be useful to distribute it to participants before the meeting so that they might use it in preparing their contributions. However, I will want back from these participants data on how helpful my questions were to their efforts. You see, I am trying to get some case studies of the effectiveness of ignorance in finding and dealing with problems in requirements. It is recognized that not all problems found in a requirements specification are errors; some may be intentional omissions, design freedoms, etc.

We can imagine several reasons why a workshop participant might have overlooked an issue that I raised in my comments and questions:

1. The issue is not relevant to the part of the case study used by the participant.

2. The issue was not a problem to the participant; that is, the participant understood the specification’s intent and acted upon it.

3. The participant never intended to follow the specification

I use the general term ‘‘issue” for any of these issues that may be found, regardless of its classification.

The deal under which I will give you this document and its ASCII version is the following. You are required to have already done at least one draft of your own contribution to the workshop. If you agree to keep track of the number, severity, effect, cause, and classification of issues found in your own contribution as a result of reading my ignorant notes, then you can have my document. When you are finished, please send me a message showing me a list of all of the issues that you found as a result of your use of my comments and questions. This list should give for each issue its severity, its effect, its cause, and its classification. Please also include any comments you might have on the experience of applying my ignorant questions.

If you are interested, please contact me at dberry@csg.uwaterloo.ca .

See

%A D.M. Berry
%T The Importance of Ignorance in Requirements Engineering

%J Journal of Systems and Software
%V 28
%N 2
%P 179-184
%D February, 1995

for more details about ignorance.

Enjoy!

Dan

PS: Here is a description of the PDF file that you will receive if you agree to the terms above.

This PDF document contains the original case study to which I have attached notes representing my ignorant questions. For those who do not have access to an Acrobat Reader or who prefer to use ASCII text, there is a textual version of the specification with the notes added. The original text is marked ">" at the beginning of their lines and the notes are marked differently.

The blue notes contain corrections to the English. Sometimes, the act of changing the English in a particular way disambiguates an ambiguity and disambiguates it in a way not intended by the specifier. In the ASCII version, these notes are marked "#" at the beginning of their lines.

The red notes contain questions that occurred to me as I was reading the specification. Some earlier questions are superseded by later questions. Some questions end up being answered later; however, then I believe that the specification should have been written in a way that it answers the question at the place the question came up. In the ASCII version, these questions have no mark at the beginning of their lines.

Managing Inconsistent Specifications: Reasoning, Analysis and Action

Bashar Nuseibeh, Department of Computing, Imperial College,
London SW7 2BZ, UK
ban@doc.ic.ac.uk

In practice, inconsistency is inevitable in all real large-scale specifications. Living with inconsistency during evolutionary development is a fact of life. Therefore, there is a need to develop formally sound techniques and practical tools that ‘tolerate’ inconsistency by allowing continued reasoning and action in the presence of inconsistency. Specifically, we propose some techniques for analyzing inconsistent specifications, for analyzing the impact of different development actions on specifications - whether these actions handle specific inconsistencies or initiate some evolutionary change, and for providing automated guidance and support for acting in the presence of inconsistency.

Lightweight Validation of Natural Language Requirements

Vincenzo Gervasi, Universita' di Pisa, Pisa – Italy
Bashar Nuseibeh, Imperial College, London - UK

We report on our experiences of using lightweight formal methods for the partial validation of natural language requirement documents. These experiences support our position that it is feasible and useful to perform automated analysis of requirements expressed in natural language.

While it was not our aim to validate any particular specification, we did identify several errors in the NASA requirements for the Pressure Monitoring System on the International Space Station, that we used as a case study. Independent review by NASA also uncovered most (but not all) of the same errors.

We describe the techniques we used, the errors we found, and reflect on the lessons learned.

Eliciting Requirements from Scenarios the CREWS-SAVRE Way

Neil Maiden, City University, London

This presentation will present an overview of the CREWS-SAVRE approach to eliciting and validating system requirements with scenarios and use cases. Scenarios can be effective for eliciting requirements from stakeholders. Scenarios offer visions of future system behaviour which can be simple to communicate, explore and quick to change in response to feedback. They are "middle-level" abstractions, less formal and complete than system specifications, but more broadly applicable to explore normal and abnormal situations.

So why are scenarios not a software development "silver bullet"? One reason is that there are few systematic processes to follow. Developers rarely know how many scenarios to produce, what the content and structure of these scenarios should be, and how they should use the scenarios elicit requirements from users. As a result, most software developers currently use scenarios in an ad hoc, non-optimal way. I shall present simple-to-use processes, methods and software tools to generate and use scenarios more systematically, and hence effectively.

The CREWS-SAVRE method and software tool has been developed as part of the ESPRIT IV's 21903 'CREWS' project as a response to industrial requirements engineering needs. Features of CREWS-SAVRE include: a language for specifying use cases; automatic generation of scenarios from use cases; automatic generation of scenario alternative courses; guided scenarios walkthroughs; patterns for automatic scenario-requirement cross-checking; compatibility with Rational Software's commercial RequisitePro requirements management software tool; integration with CREWS-ECRITOIRE, which takes a structured, natural language use case, checks it for completeness, then parses it to produce a use case specification in CREWS-SAVRE.

Analysis of SCR Specifications Using Decision Procedures

Ramesh Bharadwaj, Naval Research Laboratory, Washington, DC 20375-5320
ramesh@itd.nrl.navy.mil

In recent years, model checking has emerged as a remarkably effective technique for the automated analysis of descriptions of hardware systems and communication protocols. To analyze software system descriptions, however, a direct application of model checking rarely succeeds, since these descriptions often have huge (often infinite) state spaces which are not amenable to the finite-state algorithms of model checking. More important, model checking is rarely needed to verify most properties. For software, therefore, theorem proving affords an interesting alternative. Conventional theorem provers, however, are either too general or too expensive to use in a practical setting (in terms of the required level of user sophistication, human effort, and system resources). To be more useful in practice, a theorem proving system should be completely automatic, and require little sophistication on the part of its users. Additionally, in contrast to conventional theorem provers which provide little or no diagnostic information when a theorem is not true, an industrial strength prover should provide counterexamples along the same lines as model checkers.

In this talk I describe Salsa, a tool for the analysis of system descriptions written in a language based on the SCR tabular notation called SAL (the SCR Abstract Language). Salsa's core verification engine is an unsatisfiability checker (UC) which, given a logical formula, determines whether it is false. If the formula is not false, UC provides a counterexample along the lines of model checkers. UC serves as the engine for the Invariance Checker (IC) of Salsa, which may be used to check a SAL specification for unwanted nondeterminism and missing cases, and to verify the invariance of properties formulated by users. Salsa's Invariance Checker handles specifications that are too large for model checkers to analyze. This is because an induction proof -- which forms the core of Salsa's Invariance Checker -- corresponds roughly to a single pre-image computation, in the chain of computations carried out by a model checker during the generation of the fixed point. Moreover, for SCR specifications, we never ran across a property whose proof required the computation of the fixed point -- an induction proof (suitably strengthened with automatically generated invariants) always sufficed.

A Salsa prototype is currently implemented in Standard ML. We are working on a Java implementation of Salsa. Planned extensions to Salsa include the addition of decision procedures for the rationals, the congruence closure algorithm for reasoning about uninterpreted function symbols, and special-purpose theories such as for arrays and lists. We would also like to reason about quantifiers. We are also working on a compositional proof system for SAL modules, which will allow assumption-guarantee style reasoning.

Acknowledgements: This work is supported by the Office of Naval Research. The Salsa prototype was implemented by Steve Sims.

Surfacing Ambiguity in Natural Language Requirements

Erik Kamsties, Fraunhofer Institute for Experimental Software Engineering,
Kaiserslautern, Germany
kamsties@iese.fhg.de

Natural language requirements are recognized widely as incomplete, inconsistent, and inherently ambiguous. Formal and semi-formal specification techniques have been proposed to overcome these deficiencies. Completeness and consistency of these specifications can be tackled to some degree mechanically by tools. However, as a recent study shows, ambiguity rarely surfaces during the creation of a specification. Thus - since specification techniques enforce precision - the resulting specification becomes unambiguously wrong.

Our work aims at better process support *during the creation* of a specification from informal requirements in order to surface ambiguity as opposed to simulation or review of a specification afterwards. The value of our approach is that it can be applied right from the beginning of the specification process preventing effort going in the wrong direction. We have developed fine-grained process support for the SCR requirements specification technique, however, our approach is applicable to other specification techniques as well. Furthermore, the idea behind our approach for dealing with ambiguity can be extended to a prescriptive defect detection technique for natural language requirements documents. A recent case study has shown that inspecting requirements documents for ambiguities is much more effective and efficient than inspecting or modeling informal requirements without specific support for ambiguity detection. The reason is that engineers are often not aware of the variety of possible types of ambiguities.

Our future work aims at extending our approach to other specification techniques such as the UML and evaluating it in various settings.

Design for Test – Ensuring that Specifications Guarantee Testability

Mike Holcombe, University of Sheffield, UK.

All systems and software will be subject to testing, even if there has been substantial effort in formally verifying the design or implementation in some way. The operating environment of most systems is complex and models invariably have to make unrealistic assumptions in order to provide any basis for analysis.

If system testing is thus inevitable we should recognise this at the beginning of the project.

The theme of this work is trying to identify how testing can be made more effective by considering the issue at the requirements definition stage. Apart from the usual clients and users of the completed system there are users of interim components and deliverables, those involved with quality assurance and testing. We need to consider their needs as well.

The approach taken is based on a formal foundation involving computational modelling. We use a general formalism for describing systems in a convenient way, stream X-machines, and describe the test generation algorithm derived from stream X-machine models. The interesting aspect of this approach is that it is possible to prove results about the effectiveness of the tests. These results are of the form:

> if certain conditions relating to the specification are satisfied;
> and there is a realistic estimate of the number of extra states the implementation possesses compared to the specification;
> and the implementation is constructed in a particular way from correct components;
> and the implementation passes all the tests
> then the implementation is correct (it computes the same function as the specification)

Clearly the conditions and assumptions are important. The conditions on the specification, we call them design for test conditions are concerned with controllability and observability. We can always arrange for our specification to have these properties by augmenting it in particular ways. Once this has been done it is then possible to carry out testing in a thorough way that will detect all faults providing the other conditions are met.

We consider the case study and develop a systematic method for creating an X-machine model of the system. This relates to a communication model of the architecture of the system. This approach to specification and testing is very amenable to refinement and transformation techniques which enable test sets to be derived in tandem with specifications thus providing significant savings in terms of the cost of test set generation.

References

1. F. Ipate & M. Holcombe, Tests which are proved to find all faults. Int. Jour. Comp. Math. 63, 159-178, 1997
2. F. Ipate & M. Holcombe, A method for refining and testing generalised machine specifications. Int. Jour. Comp. Math. 68, 197-219, 1998.
3. K. Bogdanov, M. Holcombe, Automated Test Set Generation for Statecharts, , to appear in Proc. FM-Trends 98 (Boppard, Germany), Springer LNCS Series.
4. M. Holcombe F. Ipate, Correct systems - building a business process solution, Springer, Applied Computing Series,1998.

Anchoring the Requirements Process on Vocabulary

Julio Cesar Sampaio do Prado Leite
Pontificia Universidade Catolica do Rio de Janeiro, PUC-Rio
www.inf.puc-rio.br/~julio

Our work in requirements has been using the idea of a requirements baseline, a complex set of representations that are in constant evolution, as it central framework. A requirements baseline is a structure which incorporates descriptions about a desired software system in a given Universe of Discourse. It is perennial. Although it is built

during the requirements engineering process, it keeps evolving as the software construction evolves.

Our baseline uses natural language based representations, since one of its major concerns is the communication with clients. Central to this strategy is the figure of a lexicon, which anchors the meaning of the terms used in the baseline. We call this lexicon, the Language Extended Lexicon.

The Language Extended Lexicon is a metamodel designed to help the elicitation and representation of the language used in the macrosystem. This model is centered on the idea that a circular description of language terms improves the comprehension of the environment.

The Language Extended Lexicon is a representation of the symbols in the problem domain language. The LEL is anchored on a very simple idea: "understand the language of the problem, without worrying about understanding the problem". It is a natural language representation that aims to capture the vocabulary of an application.

The Lexicon's main goal is to register signs (words or phrases), which are peculiar to the domain. Each entry in the lexicon has two types of description, as opposed to the usual dictionary which has just one. The first type, called Notion, is the usual one and its goal is to describe the denotation of the word or the phrase. The second type, called Behavioral Response, is intended to describe the connotation of the word or the phrase, that is, it provides extra information about the context at hand.

We plan to show how the use of our lexicon, on the Case Study for the Dagstuhl Seminar, does help the elicitation task as well as provide an anchor for other representations of the baseline. We will build a LEL based on the written information provided by the case study and will report on the problems we have faced to produce it. On the other hand we plan to show how the vocabulary produced is an anchor for other requirements representations, such as scenarios and requirements sentences. We will also present the description of the process we will use to produce the lexicon.

Approach to Support the Implementation of Requirements Changes

Antje von Knethen
AG Software Engineering, Department of Computer Science,
University of Kaiserslautern, Germany
vknethen@informatik.uni-kl.de

Most software systems have a long life time. Thus, changes to the system are unavoidable (e.g., requirements changes). Several activities have to be performed in the case of a requirements change. For example, the impact of a change on the software documentation (i.e., customer requirements, developer requirements, design, code, etc.) has to be analyzed and the software documentation has to be changed in a consistent way.

Change activities are difficult to perform in practice because of two reasons: First, the relations among the elements of a software documentation are not documented

explicitly (i.e., horizontal and vertical traceability are missing). The horizontal relations among the elements of an artifact are implicit (e.g., among operations of a class diagram and of state diagrams). Alike, the vertical relations among the elements of different artifacts are not documented explicitly (e.g., among requirements in natural language on customer requirements level and classes of a class diagram on developer requirements level). Second, the artifacts of a software documentation are not structured to locate "typical" changes easily (i.e., "typical" changes are not encapsulated).

Our approach is to develop a semantic-based model (i.e., to define the elements and relations) for horizontal and vertical traceability among the elements of customer requirements, developer requirements, and software design. Furthermore, guidelines are proposed on how to use the model and to structure the artifacts to encapsulate "typical" changes.

We started to examine "typical" changes in a certain domain (building automation) and to develop a classification scheme for these changes. We modeled the elements of customer and developer requirements with UML (Unified Modeling Language) and have investigated the relations among the elements of and between the artifacts. Furthermore, the relations between different types of changes and the different artifacts have been analyzed.
In the near future, we plan to look into the relations between developer requirements and software design. We will describe the relations among the elements of the artifacts with the help of OCL (Object Constraint Language). Furthermore, guidelines will be developed on how to use the defined relations to support the different change activities and on how to structure artifacts to encapsulate "typical" changes.

Pattern-based Requirements Capture Applied: The SFB 501 Case Study

R. Gotzhein, M. Kronenburg, C. Peper
SFB 501, University of Kaiserslautern, Germany
email: {gotzhein, kronburg, peper}@informatik.uni-kl.de

The Case Study "Light Control System" of this Dagstuhl Seminar "Requirements Capture / Documentation / Validation" calls for the application of rigorous methods to the specification, inspection, and testing of requirements. To capture the requirements of the case study, we have applied the FoReST (Formal Requirement Specification Technique) approach. This approach serves the following objectives:

- development of a precise description of the system requirements
- customer feedback on a natural language basis
- pattern-based formalisation of requirements
- provision of a starting point for the system development team
- traceability w.r.t. the original problem description

We have developed a comprehensive requirement specification for the case study using the FoReST approach, and have applied pattern-based technologies to formalize

most of the properties. The resulting documentation is of substantial size, and has proved useful to several groups of developers. Links to the different parts of the documentation can be found at

http://www-avenhaus.informatik.unikl.de/forest/EXAMPLES/DAGSTUHL/DagstuhlEnglishStart.html

including the original problem description, documentation of the FoReST approach, various perspectives of the final FoReST specification of the case study and related publications. Postscript- as well as html-versions are available, where html-versions provide online-navigation and traceability support. All documents have been produced using FoReST tools.

The final FOREST specification of the case study goes far beyond the original problem description. This is the result of extensive feedback from the customer based on intermediate FoReST specifications, and of questions that arose during the work of the analysis team. All of these questions have been discussed with the customer, and have been resolved.

Requirements Capture, Documentation, and Validation using TRIO

Angelo Gargantini
Dipartimento di Elettronica e Informazione, Politecnico di Milano - Italy
Angelo.Gargantini@elet.polimi.it

We have applied TRIO to the proposed case study. Our first goal was to specify the case study and its requirements. TRIO is a first order temporal logic augmented with temporal operators which permit to formalize the value of temporal dependents variables at several time instants. TRIO allows precise timing requirements (such as those in the case study), and its temporal operators are very expressive and have been suitable to expresss the temporal requirements in the case study. The architecture of the system has been specified using TRIO object oriented features. TRIO allows the use of classes, inheritance, genericity and other object oriented constructs and concepts. It is also endowed with an expressive graphic representation. Our second goal has been the validation of parts of the specification by means of history checking, that takes a possible history of the system checking whether they are correct or not.

A Method for Systematic Requirements Elicitation: Application to the Light Control System

Maritta Heisel, University of Magdeburg, Germany
joint work with
Jeanine Souquieres, LORIA, Unviversity of Nancy 2, France

We applied a systematic method for requirements elicitation to the light control case study. The method consists of several brainstorming steps where the vocabulary is

fixed, the requirements are stated in natural language, and events and system operations are identified. Then, the requirements are formalized, and possible interactions between them are investigated.

The method is expressed using agendas, a concept to explicitly represent software development knowledge. An agenda consists of a number of steps to be performed, and validation conditions that help detect errors early in the development process.

Executing the agenda on the light control case study revealed that the vocabulary was not used coherently, that the glossary was partially inadequate, that functional and non-functional requirements were confused, and that requirements were missing, ambiguous or incoherent.

It turned out that the method was adequate to deal with the case study. We were able to identify a coherent subset of the given vocabulary, to resolve some of the incoherences between requirements and to point out open problems that should be solved before a realization of the system should be undertaken.

The systematic approach helped us faking ignorance, as proposed by Dan Berry.

Execution of Abstract State Machines (ASMs) for the Light Control System

Joachim Schmid, Siemens, Munich, Germany

Abstract State Machines have been used to specify the case study "Light Control System". The ASM-approach allows one to specify systems at different levels of abstraction. One can start with a high level description (understandable by the customer) and refine it (defining all used functions) to a detailed version which is executable by a tool. It seems that functional programming and ASM are a good combination to achieve this goal.

AsmGofer is an extension of Gofer wich allows one to define functions and ASM-rules with the full power of functional programming. The system also supports building graphical user interfaces, so one can combine the (executable) ASM-specification with an animation showing information about the ASM-state. The tool has also been used for making Boerger/Schulte’s ASM for the Java Virtual Machine executable.

Software Requirements Specification of the Light Control System

Egon Börger, Elvinia Riccobene, Joachim Schmid
Universita' di Pisa, Pisa – Italy

We propose a rigorous model for the informally given requirements of the light control system in terms of ASMs (Abstract State Machines).

We describe the physical architecture by conditions on the signature and by definitions on the auxiliary (mostly static) functions of the machine. The operational behavior of the control system is expressed in terms of transitions rules. The resulting "ground model" can be inspected by the customer to check which interpretation of the (partly ambiguous and incomplete) informal requirements it reflects.

By refinement techniques we obtain from the ground model an "executable model". This model can be run by an ASM simulator and is useful to execute suitable scenarios defined as test cases for the requirements.

Classification of Dagstuhl Contributions

The participants that have developed or currently develop RE methods were asked for a classification of their work. The applied classification scheme was based on Pamela Zave's classification of research efforts in requirements engineering (ACM Computing Surveys, vol.29, no.4, p.315-321, 1997).

14 participants returned the questionnaire. The RE methods developed by these participants are listed below along with additional comments made on the questionnaire form:

- Achatz, Klaus
  Process automation techniques (no abstract available)
- Berry, Daniel
  Application of Ignorance (see abstract on page 7)
- Börger, Egon
  Abstract State Machines (see page 17)
- Bharadwaj, Ramesh and Heitmeyer, Constance
  SCR and the Four Variable model (see abstracts on page 4 and 11).
  Additional comments: code can be generated from SCR requirements specifications. The SCR tool can be integrated with PVS, Spin, SMV, and others.
- Gervasi, Vincenzo
  Multiple views on natural language requirements (see page 10)
- Heisel, Maritta
  Method for systematic requirements elicitation (see page 16)
- Hoffmann, Matthias
  Tool-based requirements management of informal requirements by e.g., DOORS, RequisitePRO, etc. (no abstract available)
  Additional comments: the approach allows for defining metrics, ease to find relevant information, and ease to add project members.
- Holcombe, Mike
  X-Machines (see page 12).
  Additional comments: The approach allows for proofs of the test effectiveness. A book "Correct Systems" is available from Springer. Presumption of the approach is that a domain analysis has been made.
- von Knethen, Antje
  Traceability Approach (see page 14).
  Additional comment: the proposed approach addresses also the problem of support for requirements changes.
- Leite, Julio
  Language Extended Lexicon (see page 13)
  Additional comment: the proposed approach addresses also the problem of change management.
- Maiden, Neil
  CREWS-SAVRE method (see page 10)
  Additional comment: the proposed approach helps also to get more complete requirements.
- Nuseibeh, Bashar
  View points and inconsistency management (see page 9)

- Parnas, David
  4-Variable-SCR-Method (no abstract available).

The following table shows the detailed classification of RE methods made by the participants.

Problems that are solved by a method, status of the method, and presumptions of the method	Achatz	Berry	Börger	Bharadwaj	Gervasi	Heisel	Hoffmann	Holcombe	v.Knethen	Leite	Maiden	Nuseibeh	Parnas
Problems in elicitation and description of customer requirements													
Understanding the context of software	✓	✓		✓		✓				✓	✓		
Overcoming communication barriers			✓	✓	✓	✓	✓			✓	✓		✓
Allocating requirements among the system and the environment	✓		✓	✓		✓		✓		✓	✓		✓
Converting vague goal to specific properties or behavior of the system			✓	✓		✓		✓					✓
Understanding priorities	✓		✓	✓				✓					
Making customer requirements measurable					✓			✓					
Estimating cost, risk, and schedule							✓						
Problems in Specification of System Requirements													
Reconciling and integrating different views	✓		✓	✓	✓	✓		✓		✓		✓	✓
Evaluating different alternatives for the satisfaction of user requirements			✓	✓	✓						✓		
Obtaining complete, consistent, and unambiguous specification which can serve as basis for design	✓		✓	✓		✓		✓					✓
Making system requirements measurable					✓			✓					
Validating system requirements against customer requirements			✓	✓				✓			✓		✓
Problems in usage of the system specification during software development													
Identification of defects in the design specification			✓	✓	✓	✓							
Identification of defects in the system code			✓	✓									✓
Enabling traceability of requirements	✓		✓	✓	✓	✓	✓		✓	✓			✓
Developing a design specification	✓		✓	✓		✓		✓					✓
Deriving test cases			✓	✓				✓		✓	✓		✓
Deriving test oracles			✓	✓				✓					✓
Status of the method													
Applicable in industry	✓	✓	✓	✓	✓		✓	✓		✓	✓		✓
Supported by tool	✓		✓	✓	✓		✓	✓		✓	✓		✓
Applicable in research environments			✓	✓	✓	✓		✓		✓	✓		✓
Usable by others (e.g., user manual exists)			✓	✓	✓		✓	✓		✓			✓
Context of the method													
Applicable in every RE process	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓
Dependent on specific methods solving other problems								!			!		

Remarks:

- Originally, the form was designed for distribution *before* the seminar as a means to facilitate discussions *during* the presentation of participants. It turned out that it is not useful as questionnaire for self-estimation afterwards, because most terms used in the questionnaire allow for various interpretations, e.g., ‘context’.
- Furthermore, a checkmark has a different meaning depending on the participant answering the questionnaire. It can mean for instance “solved”, “addressed”, or “enables a solution”. This is because some problems mentioned in the questionnaire cannot be solved completely such as “overcoming communication barriers”.
- We would recommend two approaches for future work on characterizing research efforts in RE. Either allow for room for discussions on terminology and on self-estimations (e.g., “can you explain *why* do you think that your approach solves the problem of overcoming communication barriers?”) or use a more solution-oriented schema that can be interpreted in a more uniform way without discussions (e.g., which types of requirements can be specified: structural, functional, behavioral requirements).

Working Group Reports

Integrating Process, Tools, and Formal Methods

Constance Heitmeyer, Naval Research Laboratory, Washington, DC (with input from Jo Atlee, Ramesh Bharadwaj, Mats Heimdahl, Mike Holcombe, and David Parnas)

Problem: given an **approach** to specifying requirements, including a set of **tools**, describe a **process** for building a **requirements specification.**

Framework: Four Variable Model

Approaches	**Tools**	
• SCR • ASMs • RSML • X-Machines • TRIO • UML? • ...	• SCR* • McMaster TTS • Ontario Hydro • RSML tools • Tablewise • TRIO Hist. Checker • ASM tools • STATEMATE • Rational Rose • B-Tools	• Theorem Provers • Model Checkers • Consistency Checkers • Test Case Generators • Symb. Math. Tools • MATLAB • Simulators • Invariant Generators • Code Generators • Slicers/Dep. Graphs • DOORS

Criteria For Evaluating A Requirements Specification

FUNCTIONAL

- any implementation that satisfies the spec will work
  - i.e., the implementation is acceptable to the customer
- any implementation that works will satisfy the spec
- *"as simple as possible but not simpler"*

NON-FUNCTIONAL

- easy to understand
- easy to maintain
- organized for finding information quickly
- interpretable (i.e., executable)
- costs less to produce than it is expected to save
- trustable
- provides enough information to design for ease of change

Process For Constructing A System Requirements Specification

1. Identify and describe the controlled quantities by how they can be measured (e.g., range of values, etc.).
2. Identify and describe the monitored quantities by how they can be measured.

3a For each controlled variable, describe the required relation between the monitored and controlled variables.
   - to make the spec concise, use mode classes,
     i.e., functions of the history of the monitored quantities

3b. Specify timing and/or accuracy constraints on the controlled quantities.
   - e.g., "the supplies need to fall 10 feet from the specified location"

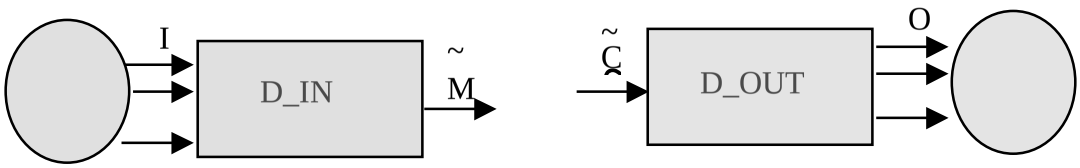
4 Specify system requirements that cannot be stated in terms of monitored and controlled quantities, e.g., weight and other physical characteristics, budget, schedule, reliability (MTBF), availability, likely system changes, etc.

Process For Constructing A Software Requirements Specification

1. Specify the IN and OUT relations.

That is, specify the device interfaces

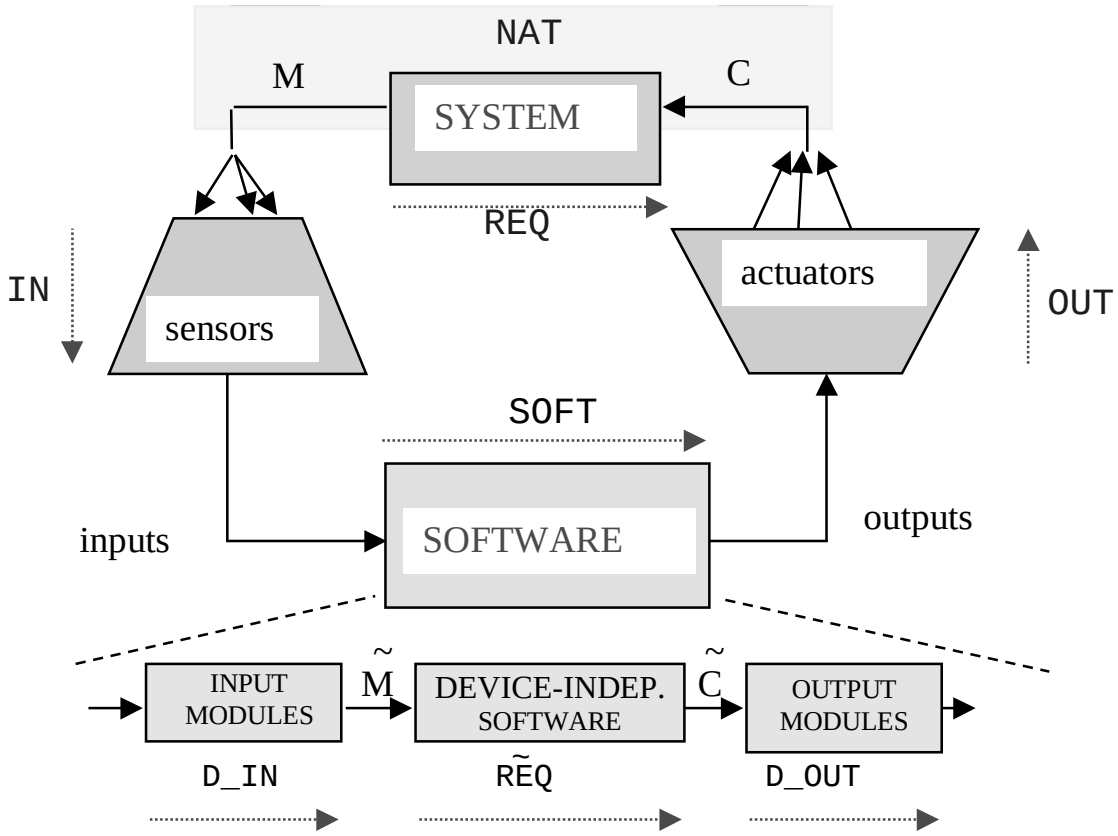
2. Specify D_IN and D_OUT.



D_IN: required relation between inputs and estimates of monitored variables
D_OUT: required relation between estimates of controlled variables and outputs

3. Specify the REQ relation.

4. Specify software requirements that cannot be stated in terms of inputs and outputs, e.g., likely changes to the I/O devices.



Verification And Validation Of The System And Software Requirements

Verify and validate the individual pieces

- REQ
- D_IN
- D_OUT

Verify and validate the end-to-end system behavior

- Are the (estimates of) controlled quantities delivered on time?
- Do the estimates of the controlled quantities satisfy the accuracy requirements?

Summary

- Given an approach (e.g., ASM, SCR, or RSML) and software tools for developing requirements, we have defined a process for constructing a requirements specification based on Parnas' Four Variable Model.
- The requirements specification consists of two parts
  - a system requirements specification
  - a software requirements specification
- The process is an idealization of the actual process that may occur in practice
  - The steps in the process may take place in a different order.
  - One may use the process to build some parts of the specification first and then go back and repeat the process to produce other parts.

A Rational Process For Building System And Software Requirements: How To Fake It!

The Light Control System Case Study

The group, consisting of 13 participants all of who in some way or the other had worked on the proposed case study before coming to the seminar, met on Wednesday, June 16, for three hours in the morning and three hours in the afternoon. The discussion focussed on clarifying the following questions:

1. What is the content of the work which has been done using the various methods for the case study?

2. What was the most problematic feature encountered during the work on the case study and which feature of the used method fits the case study best?

3. Quantification of effort and benefit of the work on the case study.

A more detailed discussion then centered around questions about the customer feedback, the traceability, the process model, and the tool support.

The detailed answers to the questions are largely available through the abstracts of the authors who presented a talk to the seminar (see the abstracts before). We try to give a short resume here. To this purpose the methods which have been used to deal with the case study are indicated below (in alphabetical order), together with mentioning (in this order) the work which has been done, the documentation which has been provided (in number of pages), the time it took to do the work (in person days). All participants except one felt that the most problematic feature of the work on the case study was the impossibility of having a feedback from a customer, given the lack of precision and of consistency of the informal requirements.

Agenda: first draft of a specification for offices after one feedback from the customer, analysing the interaction between requirements and finding missing requirements. 20 pages, 10 days.

Abstract State Machines (ASM): complete abstract specification (ground model which separates the physical environment from the control and is structured for foreseeable requests for change) and a refined executable specification. 20 pages, 8 days, tool: asmgofer simulator.

FOREST: complete (reusable and structured) specification which minimizes the gap between customer and designer, with final feedback from and acceptance by the customer. 150 pages, 40 days. Tool: editor and document creator for different views.

Ignorance: list of questions before feedback from the customer, easening the definition of a lexicon and offering help for rapid prototyping. 7 pages, 2 days.

Language Expanded Lexicon (LEL): partial lexicon (triggering more information for a satisfactory complete lexicon) and 1 scenario, defined before feedback from the customer. 25 pages, 4 days. Tools: lexicon editor, scenario editor, crc-card generator from lexicon and scenarios.

Natural language Parsing (NLP): incomplete first draft specification. 15 pages, 1 hour. Tools: text generator, graph representation and consistency checker.

SCR (USA): four-variable-model, designed for ease of change, for 1 office including fault tolerance, 5 pages, 3 days. Tool environment.

SCR (CND): specification for 1 office including malfunction and priorities. 8 pages, 5 days.

Statemate: almost complete executable specification for one office. 10 pages, 3 days. Tool: Statemate.

TRIO: almost complete executable specification for one office. 20 pages, 5 days. Tools: model generator and history checker.

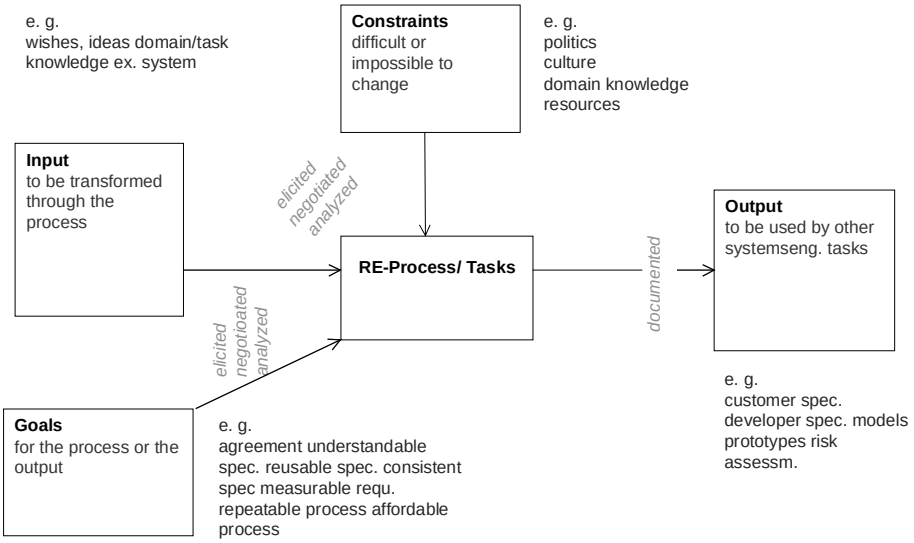
X-machines: Analysis of requirements with respect to an abstract model, series of refinements including design for test issues (test set generation). 18 pages, 5 days. Tool: test generator.

The Richness of the Requirements Engineering Process

Barbara Paech, FhG IESE (with input from Stefanie Lindstaedt, DaimlerChrysler)

The Scope of the RE-Process

The participants of the working group had quite diverse backgrounds and interests. Therefore, we started out by creating a common view on the RE-Process. The result of this effort is captured in the following picture:



The RE-process (and each individual task within that process) is driven by inputs, goals, and constraints and produces some output. The driving forces can be distinguished into forces on the solution space which are transformed through the process (inputs, e.g. customer wishes, existing systems), forces on the solution space which are impossible or difficult to change (constraints, e.g. politics, system type) and forces on the RE-process and its outputs (goals, e.g. agreement, reusable process, consistent specification). The domain knowledge, for example, is partly input (like user tasks knowledge) and partly constraint (like physical laws). All these driving forces are typically not evident at the beginning of the process, but have to be elicited, negotiated and analyzed within the process.

Depending on the forces and the required output, RE-processes have quite different characteristics. So, a process starting with a vague (product) idea of the customer and aiming at a detailed customer specification will typically be a mutual learning process between customer and developer, dominated by elicitation and negotiation tasks, while a process driven by a quite detailed customer specification (developed by the customer or in a separate project) aiming at a developer specification is typically dominated by analysis. Typically, a contract separates these two types of processes. However, even without a contract there is an important kind of border between the two processes due to the effects of change. Change of requirements is easier in the

first part than in the second. Typically the detailed customer requirements reach a state of complexity and settledness that changing one or several ones of them requires a lot of effort, time, and money. Thus, on that border the willingness of the RE-participants to accept change decreases.

Examples of Industrial RE-Processes

Based on this common understanding we looked at specific instances of industrial RE-processes and their problems. One problem we discussed for several hours was the situation of pre-development and series-production groups at DaimlerChrysler. A pre-development group typically gets a new, vague idea from research (e.g. avoid sliding of the car when braking). The task of pre-development is then to build a first prototypical system which can work in a car. Afterwards the prototype is given to a series-production group. The task of this group is now to re-implement the system cleanly and to consider the additional constraints the system has to meet like specific control units used, etc. The problem is, that much of the design rationale of the prototype is lost when it is given to series-production. So the engineers basically have to re-engineer the prototype. The question is now: how is it possible to capture the important design decisions for reuse in series-production without hindering the creativity and spontaneity of the pre-development engineers who develop the prototypes?

During the discussion 4 ways of dealing with this problem surfaced:

1. The series people could elicit the design rationale from the pre-development group after the prototype has been constructed.
2. The pre-development people should document each design decision during the prototype development.
3. Mix people from pre-development and series-production: send some people from pre-development with the prototype to series-production, and /or have some series people already involved in the pre-development.
4. Combine all three approaches: people in pre-development collect some notes on their rationale, reflection workshops are held periodically in which pre-development tries to make their rationale and experiences explict. On such workshops people from series-production might be present(especially when the prototype becomes more and more mature) and should document the knowledge elicited. The intend is to use the pride of the pre-developers to motivate them to talk about their experiences. The whole process of the reflection workshops should be driven by the informational needs series-production has.

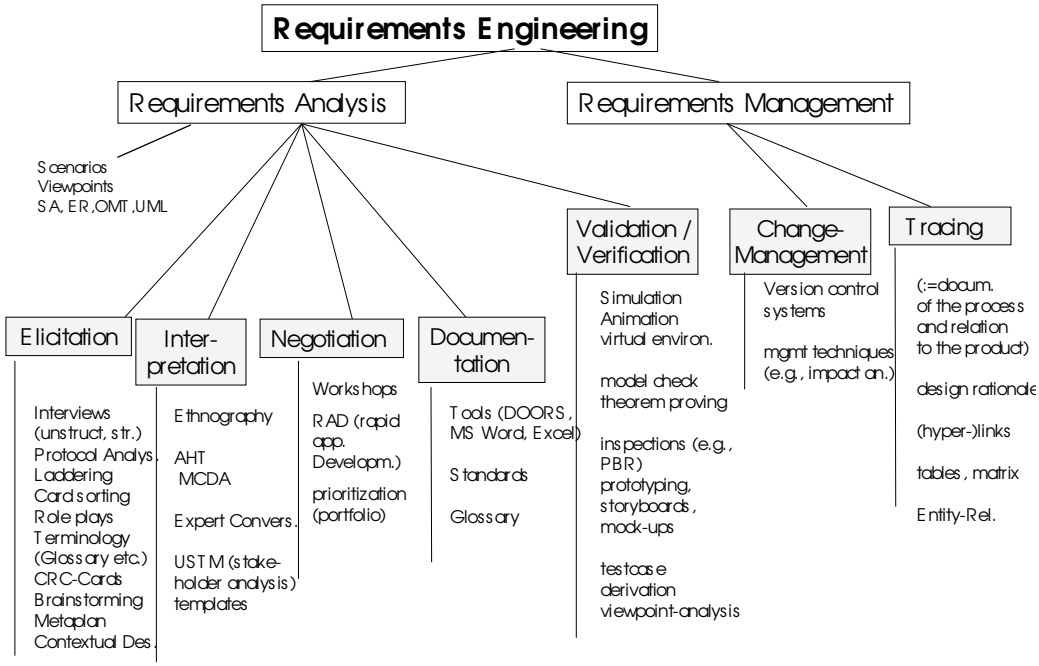
Obviously, options 1 to 3 by themselves are not feasible. However, the combination of them described in option 4 (which could be varied in several ways) tries to combine their merits in a most helpful way.

Another pressing problem DaimlerChrysler faces is the question of how to deal with quality requirements. Already in his presentation on the first day of the seminar Kurt Schneider (DaimlerChrysler) illustrated how quality models can be used. However, this approach is not entirely satisfying and the hope was to get pointers to research work dealing with this problem. Discussing this topic it became apparent that there is no ready to use approach available. In order to break typical quality requirements like usability and reliability down into testable requirements a lot of domain knowledge is

required. This knowledge needs to be collected and reused so that it can already be applied in early RE-process phases like in the creation of the contract.

RE-Techniques

Having explored the problem dimension of RE-processes we started on the second day from the solution viewpoint and looked at different techniques applicable during the RE-process. The following picture shows a quite impressive list of techniques for the different RE-tasks which, however, is far from being complete. It just reflects a quick brainstorming of the participants.



We tried somewhat harder to identify techniques for handling non-functional requirements, but besides general techniques for capturing, structuring, and documenting knowledge (in particular quality features) and on resolving conflicts between them nothing much seems to be around. A conjecture was made that the reason for this is the general attitude of treating non-functional requirements as being secondary to the functional ones. The discussion on techniques also made clear that the most fundamental skills of a requirements engineer are communication skills.

The picture also shows that some techniques are more general than others - the most general ones are listed separately. These techniques can be applied to many tasks, but they still have aspects which make them particularly suited for a specific tasks. So, e.g. scenarios are certainly a way to document requirements, but their main benefit is in the elicitation tasks. Also, analysis methods like SA are mainly sold as a documentation means, however, there are specific usages of these techniques for e.g. elicitation purposes.

While it is quite easy to give a rough categorization of the techniques according to the RE-tasks and the forces mentioned above, we found it quite hard to describe the

situations in detail in which a particular technique is of most benefit. Thus, it is difficult to give general recommendations for techniques to industrial requirements engineers, since the choice depends on many implicit factors.

RE-Process Improvement

Comparing this result with the result of the problem discussions from the first day, it seems that improvement of industrial RE-processes is achieved by first designing an appropriate organizational solution and only then filling out the technical details based on the specific factors of the company.

The working group ended with a characterization of the major achievements of the RE-community in the last years. The participants felt that important points are the focus on elicitation, the shift from requirements modeling to requirements design and the overall increasing recognition of the importance of RE in industry.

Also, it was emphasized that Requirements Engineering and Software and Systems Engineering are getting reunited in several ways:

- there is an integration of architecture and design issues in the requirements specification,
- a lot of S(W)E-tasks which one would normally not view as RE-tasks (e.g. evaluating project descriptions) contribute to the requirements engineering process and benefit from RE-techniques,
- a lot of outputs from the RE-process can be used for other S(W)E-tasks than only design (e.g. using scenarios for test case generation),
- requirements engineering is a continuous process within the S(W)E-process and does not end with the delivery of the software system created.

Altogether, the group agreed that there is a rich set of issues and techniques associated with the RE-process becoming more and more valuable for systems and software engineering as a whole.

This report was compiled by Erik Kamsties, Fraunhofer IESE, Germany.