Chapter 13

BRIDGING THE REQUIREMENTS TO DESIGN TRACEABILITY GAP

How an integrated decision model helps closing the gap

Bernhard Turban[a], Markus Kucera[b], Athanassios Tsakpinis[b], Christian Wolff[c]
[a]*Electronic Systems Engineering, MBtech Group, Neutraubling, Germany, bernhard.turban@mbtech-group.com;* [b]*Competence Center SE, University of Applied Sciences, Regensburg, Germany, {markus.kucera,athanassios.tsakpinis}@informatik.fh-regensburg.de;* [c]*Media Computing, University of Regensburg, Germany, christian.wolff@computer.org*

Abstract: Requirement traceability ensures that software products meet their requirements and additionally makes the estimation of the consequences of requirement changes possible. In this article a case study analyses symptoms of this problem in the process model of ISO 12207, the foundation of SPICE (ISO 15504), and CMMi. Our analysis is directed at deriving a concept for the integrated extension of current traceability models with the aspect of *documented design decisions*. This integrated decision model is presented along with an additional case study which illustrates the advantages of this approach for traceability.

Key words: requirements engineering, traceability, design, rationale management, decision, embedded systems, SPICE, ISO15504, ISO 12207, CMMi

1. INTRODUCTION

In the development of safety-critical embedded real-time systems, safety and reliability are of major importance[1] (cf. ISO 61508). Therefore, control and improvement of software processes (cf. ISO 15504 SPICE) are of high significance. In these processes, traceable and consistent elaboration of requirements throughout all development cycles (especially the design phases) is mandatory. However, today's document-heavy approaches face problems with redundancy and synchronization of different stakeholders' views. To handle these issues, we propose an approach that concentrates on

maintaining one consistent view of all requirements between all stakeholders. In the following design phases, the stakeholders and artifacts of the different engineering disciplines (Systems engineering, hardware (HW) and software (SW)) shall be connected by a lightweight model and tool-based traceability approach.

The core of this approach is a decision model which links requirements, design problems and design together. As a result, new constraints on the solution space can be identified and used in a similar way as requirements. Whereas former traceability approaches regarded decisions as valuable side information, in our model decisions get directly integrated in the classical traceability information forming traceable chains of decisions through the design process. As a side effect, the approach addresses several problems in rationale management and encourages direct communication between the stakeholders. This decision model has been integrated into a software development tool which acts as a bridge between requirements tools like DOORS and design-oriented tools like Matlab Simulink or Artisan Realtime Studio.

We start in chapter 2 with describing the state of the art in traceability research and continue in chapter 3 to analyze problems in establishing traceability information in current process models. This builds ground for chapter 4 which introduces our integrated decision model that helps to improve currently used traceability models. A case study shows how the model can be applied in a practical setting. Chapter 5 gives hints on the model's further support potential for designers while chapter 6 draws a short conclusion.

2. REQUIREMENT TRACEABILITY TO DESIGN

Requirements management, i.e. the activity of organizing, administrating and supervising requirements during the whole development process, and *Traceability* are mandatory actions to fulfill exigencies imposed by software engineering standards like SPICE[1] (*Software Process Improvement and Capability dEtermination*[2]) or CMMi (*Capability Maturity Model Integration*[3]). *Traceability* means "comprehensible documentation of requirements, decisions and their interdependencies to all produced information/artifacts from project start to project end"[4(p.407)]. Between artifacts or respectively models of different development processes emerging structural interruptions and semantic gaps[5,6,7(p.138f)] endanger a project's consistency and the common understanding of its stakeholders. Traceability

[1] In the following, we concentrate on SPICE, but our claims are equally valid for CMMi, as both process models are based on the process model of ISO 12207.

relationships are intended to close these gaps. Paech et al.[8] indicate that traceability in relation to the design of artifacts is typically seen as a set of bidirectional relationships between requirements and their *fulfilling* design entities[9].

Research on traceability has proposed various approaches for establishing or retrieving traceability dependencies. Rochimah et al. present an evaluation of current state of the art traceability approaches concerned with SW evolution[10]. Research has shown that manual creation and maintenance of traceability relations requires enormous effort and includes substantial complexity[11,9,12]. The study of Rochimah et al. further shows that current research on traceability focuses on automating traceability link generation[10(Table 4)]. Some automation approaches still depend on manually established links that are then enriched by supporting automation mechanisms while others are fully automated. We have analyzed the scope of automation of these approaches and identified two major areas of automation:

- Finding interdependencies between different requirements artifacts (e.g. textual documents, use case descriptions, feature-models or analysis models) concerned with requirements.
- Finding interdependencies between design and code artifacts.

Only the approach suggested by Spanoudakis[13,14] tries to establish automated trace links from requirements to models, focusing on analysis models, though.

It is striking that current automated link generation approaches do not concentrate on establishing links between the requirements world and the design world. We believe this can be explained by the “name mapping” or “name referencing” phenomenon: Instead of creating explicit links between items, the same names are used[15(p. 224)].

If no automatic code generation is available for a design tool and code must be typed manually, traceability must also be established between design and code. As design is (and should be) a more abstract view on the problem modeled, traceability can also be established by naming corresponding elements in design and code identically. This is an explicit heuristic. In addition, another heuristic significantly reinforces this effect in an implicit way: It is very important to achieve a common understanding of the project for all different stakeholders. This can only be achieved, if the project develops a common vocabulary for its used terms. Therefore, in the field of requirements specification, using precise terminology and establishing adequate terminology management is a central principle.

However, concerning traces from requirements to design, Paech et al.[8] point out that these relationships can be of a more complex nature (cf.

Fig.13-1 below). In principle, *non-functional requirements (NFR)* restrain *functional requirements (FR)* and *architectural decisions (AD)*. On the other hand, NFRs are realized by FRs and ADs, whereas FRs are realized and restrained by ADs. Egyed et al. discuss similar observations[11] where they map FRs to nonfunctional aspects (or *software attributes*) where they identify conflicting and supporting situations. It becomes clear that such dependencies are highly dependent on the design context (e.g. the potential conflict can also be nonexistent, if a FR and a nonfunctional aspect are realized in different components).

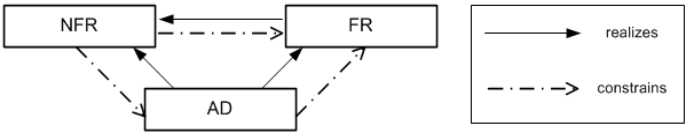

Figure 13-1. Relationships between non-functional (NFR), functional requirements (FR) and architectural decisions (AD) according to Paech et al.[8].

Tracing requirements from the original requirements specification to design by simple bidirectional links is inaccurate as this would assume the transition from requirements to design to be a fairly linear and one-dimensional process. We rather believe that this transition is a creative and complex mental transfer process performed by designers when gradually transforming the problem space into a solution space (so called *Wicked Problems*[16]). Thus we assume a substantial gap between the world of requirements and design (resp. code), since requirements represent the problem world, whereas design forms the solution world. Accordingly, we believe that (automatic) traceability link generation can be a valuable support mechanism to find dependencies between *within* each sphere (e.g., finding all references of a variable used in source code is a simple and state-of-the-art feature), but it faces high barriers when trying to bridge both worlds. It can be agreed with Egyed et al. that "while some automation exists, capturing traces remains a largely manual process" [17(p.115)] and such links degrade over time and must be continuously maintained. Further, the type of usage of the link information must be considered: Egyed et al.[17] distinguish between *short-term utilization* (are all requirements considered?) and *long-term utilization* (assessing a particular change years later). Short-term utilization is more or less covered by the simple link concept usually applied by today's traceability understanding, whereas for mid- and long-term utilization of more complex relations additional information such as decisions and their rationale must be considered.

3. SHORTCOMINGS OF CURRENT PROCESS ARTIFACT MODELS

The SPICE process model uses the standardized process model of ISO 12207[2]. This process model demands the following artifacts:

- A system requirement specification (SYS-RS) collects all requirements retrieved from the user by the user requirements specification.
- The SYS-RS builds the basis for a high-level system design model with the prior emphasis on HW-SW-partitioning.
- A HW requirements specification (HW-RS) for the HW and a SW requirements specification (SW-RS) are derived from the SYS-RS and the system design model.
- The HW-RS and SW-RS are the basis for the corresponding HW and SW design models.

We present a detailed analysis of the problems encountered applying traceability to this kind of process model in [18]. In embedded development, requirements concerning the system, SW and HW are strongly interwoven and thus a clear separation between requirements and design artifacts leads to high redundancy and cluttered information. The following example will demonstrate this (a detailed discussion can be found in [18]).

The example has a system requirements specification (SYS-RS) with three requirements causing a problem encountered in our practice context:

- Req.1: An external watchdog component must monitor the system.
- Req.2: Parametric data must be changeable by the customer during operation.
- Req.3: Parametric data must be stored in *Electrically Erasable Programmable Read Only Memory* (EEPROM).

In current practice, the system design determines that the system will include a microcontroller, an external watchdog component and an external EEPROM (cf. Fig.13-2). The HW requirements specification (HW-RS) derived from the SYS-RS and system design again contains *Req.1* and *Req.3* linking back (fat upward arrows in Fig.13-2) to the SYS-RS. The detailed HW design determines that watchdog and EEPROM will share the connection pins to the controller by a *Serial Peripheral Interface* (SPI) – communication interface, because other connected components have already used up all remaining pins of the controller. Req.1 gets linked to the watchdog symbol and Req.3 to the EEPROM symbol in the HW design.

The SW requirements specification (SW-RS) contains Req.1, Req.2 and Req.3 linking back to the SYS-RS. During SW design, the architect discovers the potential resource conflict in the shared usage of one SPI for EEPROM and watchdog. Since driving the EEPROM is very time intensive and triggering the watchdog is time critical, the architect rates this

combination as a risk, but changes of the HW are rejected due to potentially higher costs. The solution for this conflict, the EEPROM and watchdog drivers must be “artificially” coupled to implement a cooperative handshake solution (Fig.13-2: association in SW design model marked “!!!”). The solution implies that the planned original standard drivers of a supplier must be adapted internally. In the further progress of the project, these adaptations cause extra efforts not traceable to its background.

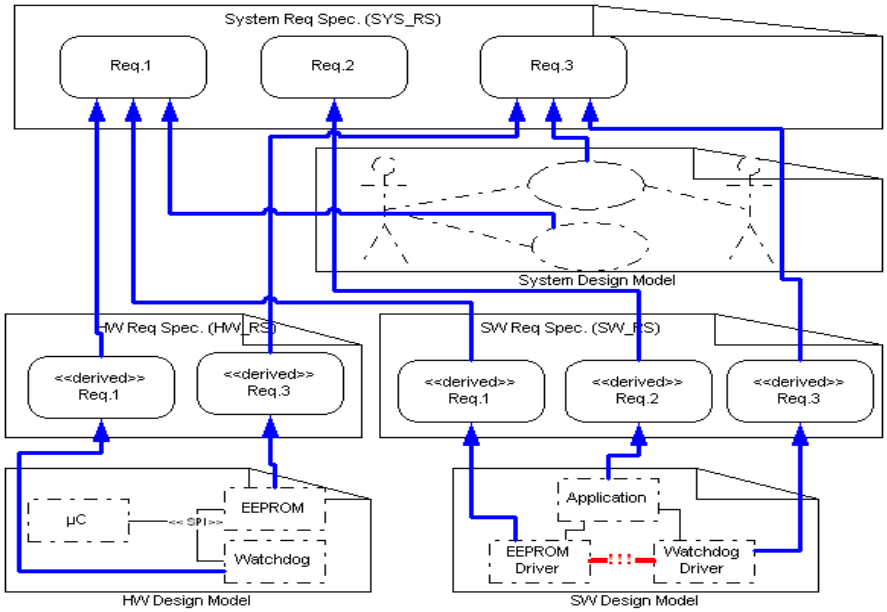

Figure 13-2. An example following previous approaches

4. AN INTEGRATED DECISION MODEL

The above example illustrates two central problems: First, the requirements in HW_RS and SW_RS are copies of the requirements in the SYS_RS, leading to high redundancy. In many cases, SW or HW functionality is already clearly demanded for in the user requirements specification. Thus a clear separation of those requirements must be taken over into the SYS_RS and SW_RS respectively HW_RS causing additional effort and redundancies. To avoid this, we propose to use a single central requirements specification containing one consistent view on all aspects of the system to be developed. When a current state of the art requirements

management tools like DOORS® is used, a HW-SW-partitioning of requirements is also viable using attributes (proposed values: System, HW, SW, construction, management). Thus, HW-RS as well as SW-RS can be derived as views with a filter on the specific attribute value.

Second, design activities concerning one design artifact (in our example HW design) can have serious implications for other requirement or design artifacts (in our example SW design). This fact is partially considered in the process model of SPICE: System design has high impact on its SW design by raising new "requirements" in addition to the original requirements of the stakeholders. Thus, the idea behind a SW_RS is to collect the SW-related requirements from the SYS_RS and derive new requirements from the System design together. However especially in the automotive sector, SW-design must be subordinated under constraints of extremely cost-optimized HW components. At the moment, SPICE neglects these critical connections between HW and SW.

4.1 Introducing the integrated model

Another issue in SW requirements which might benefit from more intensive discussion is their negotiability. "Real requirements" are part of the contractual basis between the stakeholders in a project. Changes of such "real" or "contractual" requirements must typically be harmonized with the customer via a *Change Control Board* (CCB) or a similar body used in project management. For requirements resulting from design decisions (modeled as *DesignConstraints* here, see below), it is possible to search for a project internal solution first, before escalating the issue to the CCB is considered. Thus, both kinds of requirements should be strictly separated in their notation.

For this, we propose to use the following taxonomy (Fig. 13-3) to support a more explicit distinction:

- *Requirements* are directly allocated to the SYS-RS, since they concern the legal agreement between customer and contractor.
- „Requirements" derived from requirements or designs are called *DesignConstraints.*
- *Requirements* and *DesignConstraints* have similar qualities and structure. Thus, we use the term *RequirementalItem (RI)* for both items.

Requirements have to refer to their origin[7,4]. This relation should apply to all *RIs*. The origin of *DesignConstraints* lies in previously made design decisions solving the conflicts/forces between *RIs* and/or architectural items constraining the broader more abstract solution space to a more concrete one. These considerations lead to our idea of directly integrating a *decision model* into traceability information (cf. Fig.13-4) helping to document the

origin of new *DesignConstraints* (this especially helps to make the HW' s influence on SW more transparent[19(p.415)]) in a lightweight and need-oriented way. Fig.13-4 shows this concept extending today's traceability models[8] by an *explicit decision model*. The diagram sketches a concrete situation, where a conflict between two requirements (*Req_1*, *Req_2*) and two UML model elements (*Class1*, *Class2*) is resolved by a *design decision* resulting in two new *DesignConstraints* (*DesConstraint1*, *DesConstraint2*).

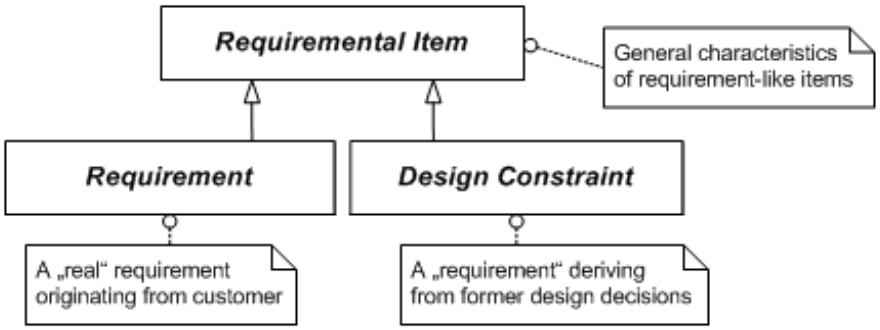

Figure 13-3. Requiremental items taxonomy

The conventional scheme of relating requirements to realizing model elements is extended by a dialog allowing the capturing of documented decisions. In this dialog, elements of the requirement model and the design model which are conflicting or which cause a problem can be chosen. Equally, diagrams describing aspects of the conflicting situation can be attached as additional information (<<*documenting diagrams*>>).

Furthermore, the decision can be specified on demand via a text component. The text component accepts unstructured text, but may also provide adequate description templates to support the decision documentation. A possible way for structuring this text is shown in Fig.13-4 with the decision's attributes *assumptions*, *rationales* and *solution specification.*

The decision model presented here is strongly connected to the research area called *rationale management* (RM, cf. [20] for an overview). In [18(Ch.5)], we provide a detailed description of the dependencies and implications of research in RM on our decision model.

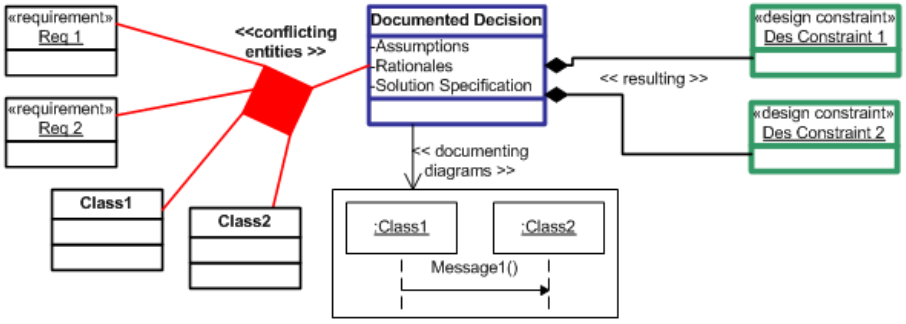

Figure 13-4. Documented decisions bridge the gap between requirements, design elements and resulting design constraints.

4.2 Applying the Decision Model

The following example illustrates how the same situation as in the example given above is solved by our proposed approach. The system design is done just as proposed in chapter 3.1 (Fig.13-5). The SYS-RS contains an attribute that allows a SW-HW partitioning. Req.1 and Req.3 are marked as relevant for HW and SW, Req.2 only for SW. The HW-RS is not directly applied, since the relevant HW requirements are marked in the SYS-RS. The HW design is done similar to Chapter 3.1 and linked to the Req.1 and Req.3 in the SYS-RS. The SW-RS is not applied, since the relevant SW requirements are marked in the SYS-RS. The SW design will be developed from the SYS-RS and the system design model. The architect discovers the same problem concerning watchdog and EEPROM. He opens a decision wizard and marks Req.1 and Req.3 as conflicting and links to the HW-design diagram that documents the conflict. As a further rationale, the architect textually documents „synchronization conflict at SPI between time intensive EEPROM application and time critical watchdog application“. A further click helps the architect to put the conflict into the risk list. In the resulting *DesignConstraint*, the architect sketches the cooperative handshake and links the *DesignConstraint* to the EEPROM and watchdog design elements in the SW design.

This decision model is currently being implemented in a traceability tool. In the further project progress necessary changes are detected early by impact analyses and the additional costs can be compared to the cost savings of the rejected HW change.

The artifacts HW-RS and SW-RS not realized can be generated out of the model, on demand by summing up all requirements related to the

corresponding design (HW design model for the HW-RS, SW design model for the SW-RS).

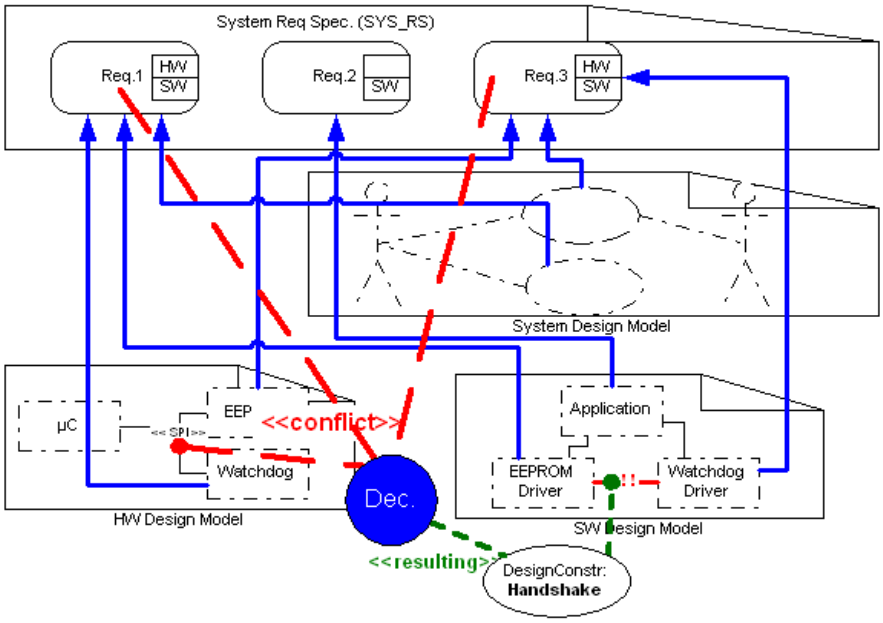

Figure 13-5. An example following our proposed approach.

The idea of including decisions into the traceability models is not new (e.g. cf. the recently introduced approach by Tang et al.[21]). In contrast to other approaches that record decisions (rationale) as additional information, our decision model directly integrates into the traceability schema by the following key characteristics:

- Conflicts between *RequirementalItems* (and design elements) can be modeled.
- Decisions do not directly influence dedicated design objects, but they bear *DesignConstraints* that can be the treated as new “requirements” (called *RequirementalItems* here).
- These *RequirementalItems* are part of all subsequent traceability processes.

For a detailed analysis on the differences to other approaches of documenting rationale in design, we recommend reading [18].

5. HOW THE NEW DECISION MODEL PROVIDES ADDITIONAL SUPPORT TO DESIGNERS

In the following, we will discuss additional connections and advantages of the proposed decision model in relation to design-related issues.

5.1 Patterns

„Patterns, as used in software engineering, constitute one of the most heavily used approaches for organizing reusable knowledge"[22(p.19)]. Patterns define the abstract core of a solution for a continuously recurring problem thus allowing to reapply the solution tailored to the concrete problem[23]. Patterns are described using a structure template. Even though different authors use slightly different templates, the description of the problem (often referred as forces), the solution and its consequences are part of all pattern templates. Our decision model can be described in terms of such a pattern template (see also [24(Table1)]): The conflict situation corresponds to the problem description part, whereas the description of consequences in a pattern description could be modeled by resulting new *DesignConstraints*. Due to this analogy, we believe our approach can provide valuable support in selecting design patterns (e.g. the conflict situation of a decision can indicate the usage of a specific pattern). At the same time, it can help knowledge engineers in identifying interesting solutions as *new patterns* (on the relationship between design decisions and patterns also refer to [24,25(p.209)]). A pattern library for decisions in modeling embedded systems could be the ultimate goal of such an effort.

5.2 Ensuring Adequate Realization of Design and Decisions

As Posch et al.[25(p.38)] underline, architects also have to ensure that their design settings are adequately considered and realized by other designers or coders. Using our model, designers can model the consequences of a decision as *DesignConstraint* and relate the DesignConstraints as new "requirements" (in our terminology: *RequirementalItem* (*RI*)) for design elements. Besides usage in further design or coding processes, the list of assigned *RIs* to a design item can also be used as basis for reviews on design and implementation of the item.

5.3 Support for Architecture Evaluation

Our approach can also provide valuable support at maintenance and evaluating architectures[26]. As Moro[27(p.321)] points out the usage of patterns

and other decisions must be documented for later maintenance and architecture evaluation issues.

6. SUMMARY AND OUTLOOK

This article shows the interdependencies between the SPICE-layered process model, requirements, traceability, designs and decisions with special attention on low redundancy in the traceability information. We suggest a strict separation between contractual mandatory requirements ("real requirements") and requirements resulting from former design decisions (*design constraints*). Design decisions are interpreted as links between requirements, designs and derived *DesignConstraints*. This closely connects and synchronizes approaches in requirement traceability and rationale management. In accordance with the literature[28,6,8,29,12], it can be argued that the influence of requirements on design processes – and vice versa – is only insufficiently modeled by bidirectional linkages.

In the course of a cooperation project between *MBtech Group* (formerly with the *Micron Electronic Devices AG*, since June 2008 part of *MBtech*), the *Competence Center for Software Engineering* of the *University of Applied Sciences Regensburg* and *the Media Computing Group* of the *University of Regensburg* a prototype system is being implemented which includes the decision model presented here. Customer workshops at *MBtech* have shown promising acceptance by designers. At the moment, the tool environment faces first practical applications in real world projects.

Acknowledgements

This research has been funded by the Bavarian Ministry of Economic Development (Grant Nr. IUK229). Furthermore, we want to thank all partners that contributed to our research.

7. REFERENCES

1. O. Benediktsson, R. Hunter and A.D. McGettrick. Processes for Software in Safety Critical Systems. In: Software Process: Improvement and Practice 6 (1), 47-62 (2001).
2. K. Hörmann, L. Dittmann, B. Hindel and M. Müller. SPICE in der Praxis, Interpretationshilfe für Anwender und Assessoren, dpunkt Verlag, Heidelberg (2006).
3. R. Kneuper. CMMI. Verbesserung von Softwareprozessen mit Capability Maturity Model Integration. Volume 2, dpunkt Verlag, Heidelberg (2006).
4. Ch. Rupp. Requirements-Engineering und –Management, Volume 2, Hanser, München (2002).

5. M. Lindvall. A study of traceability in object-oriented systems development. Licenciate thesis, Linköping University, Institute of Technology, Sweden (1994).
6. A. von Knethen. Change-Oriented Requirements Traceability. Support for Evolution of Embedded Systems, Fraunhofer IRB Verlag, Stuttgart (2001).
7. Ch. Ebert. Systematisches Requirements Management, dpunkt, Heidelberg (2005).
8. B. Paech, A. Dutoit, D. Kerkow and A. von Knethen. Functional requirements, non-functional requirements, and architecture should not be separated - A position paper, REFSQ Essen (2002).
9. O. Gotel, O., A. Finkelstein. An Analysis of the Requirements Traceability Problem. Proceedings First International Conference on Requirements Engineering 1994, pp. 94–101 (1994).
10. S. Rochimah, W. Wan Kadir, A. Abdullah. An Evaluation of Traceability Approaches to Support Software Evolution. International Conference on Software Engineering Advances (ICSEA) (2007).
11. A. Egyed, P. Grünbacher. Indentifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help. IEEE SW November/December (2004).
12. B. Ramesh, M. Jarke. Toward Reference Models for Requirements Traceability. IEEE Transactions on Software Engineering, 27(1) (2001).
13. G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause. Rule-Based Generation of Requirements Traceability Relations, Journal of Systems and Software 105-227 (2004).
14. G. Spanoudakis, "Plausible and Adaptive Requirement Traceability Structures," in Proc. 14th International Conf. Software Eng. and Knowledge Eng. (2002).
15. M. Müller, K. Hörmann, L. Dittmann and J. Zimmer. Automotive SPICE in der Praxis: Interpretationshilfe für Anwender und Assessoren. Dpunkt 1. Auflage, Heidelberg (2007).
16. W. Kunz, H. Rittel. Issues as elements of information systems. Working Paper 131, Center for Urban and Regional Development, University of California, Berkeley (1970).
17. A. Egyed, P. Grünbacher, M. Heindl, S. Biffl, Value-Based Requirements Traceability: Lessons Learned. 15th IEEE International Requirements Engineering Conference (2007).
18. B. Turban, M. Kucera, A. Tsakpinis and Ch. Wolff, An Integrated Decision Model For Efficient Requirement Traceability in SPICE Compliant Development, Fifth Workshop on Intelligent Solutions in Embedded Systems (WISES), Madrid (2007).
19. P. Liggesmeyer, and D. Rombach (Eds.): Software Engineering eingebetteter Systeme Grundlagen - Methodik – Anwendungen. Volume 1., Elsevier, München (2005).
20. A. Dutoit, A., R. McCall, I. Mistrik and B. Paech (Eds.). Rationale Management in Software Engineering. Springer, Berlin (2006).

21. A. Tang, Y. Jin, J. Han, A rationale-based architecture model for design traceability and reasoning. Journals of Systems and Software Volume 80(6) 918-934. (2007).
22. A. Dutoit, R. McCall, I. Mistrik and B. Paech. Rationale Management in Software Engineering: Concepts and Techniques. In [20 (p.1-48)] (2006).
23. E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA (1995).
24. N.B. Harrison, P. Avgerion, U. Zdun, Using Patterns to Capture Architectural Decisions. IEEE Software 38- 45 July/August (2007).
25. T. Posch, K. Birken and M. Gerdom, Basiswissen Softwarearchitektur- Verstehen, entwerfen, bewerten und dokumentieren. dpunkt, Heidelberg (2004).
26. P. Clements, R. Kazman, M. Klein, Evaluating Software Architectures – Methods and case studies. Addison-Wesley (2002).
27. 27 M. Moro, Modellbasierte Qualitätsbewertung von Softwaresystemen, Books on Demand GmbH, 1. Auflage (2004).
28. A. von Knethen, A Trace Model for System Requirements Changes on Embedded Systems, In Proc. of 4th International Workshop on Principles of SW Evolution, Sept. (2001).
29. R. Pettit, Lessons Learned Applying UML in Embedded Software Systems Design, Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, Wien (2004).