Entscheidungen bei der Erfassung nicht-funktionaler Anforderungen

Andrea Herrmann

Lehrstuhl Software Engineering, Institut für Informatik
Universität Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg
herrmann@informatik.uni-heidelberg.de

Abstract: Die Anforderungsspezifikation eines Systems ist das Ergebnis vieler Entscheidungen. Diese voneinander abhängigen Einzelentscheidungen bei der Identifizierung der relevanten Anforderungen (und speziell der nicht-funktionalen) werden in dieser Arbeit systematisch betrachtet und es wird diskutiert, wie sie bewusst und systematisch durch eine Requirements Engineering Methode unterstützt werden können, beispielsweise mit MOQARE. Eine solche Entscheidungsunterstützung erhöht die Qualität deren Ergebnisse (der Anforderungen) und damit des IT-Systems.

1 Motivation

Die Anforderungsspezifikation eines Systems ist das Ergebnis vieler einzelner, voneinander abhängiger Entscheidungen. Jede Anforderung entsteht durch eine oder mehrere Entscheidungen [RPA01]. Wir beschäftigen uns hier mit der Frage, wie man ausgehend vom „weißen Blatt“ noch nicht existierende Anforderungen identifiziert, die wichtig genug sind, um aufgeschrieben zu werden, und solche gar nicht erst aufschreibt, die unwichtig sind, d.h. wie man eine Wunschliste erstellt. In der Requirements Engineering (RE) Literatur werden vor allem diejenigen Entscheidungen behandelt, die nach der Anforderungsspezifikation zu treffen sind, beispielsweise das Lösen von Konflikten oder der Entwurf einer Lösung, die die Anforderungen möglichst gut unterstützt, aber auch Prozessentscheidungen (z.B. in [RPA01]). Diese werden andernorts bereits behandelt, u.a. von der Autorin dieses Beitrags [HP06, HPP06].

Dieser Beitrag diskutiert, wie die Entscheidungen darüber, welche nicht-funktionalen Anforderungen (NFA) das System erfüllen soll, bewusst und systematisch getroffen werden können. Eine solche Entscheidungsunterstützung erhöht die Qualität deren Ergebnisse (der Anforderungen) und damit des IT-Systems.

Kapitel 2 gibt eine kurze Einführung in die Entscheidungsunterstützung bei der Erfassung von Anforderungen. In Kapitel 3 wird dargestellt, wie MOQARE Entscheidungen bezüglich NFA unterstützt. Kapitel 4 fasst diesen Beitrag zusammen.

2 Einführung in die Entscheidungsunterstützung bei der Erfassung von Anforderungen

Eine Entscheidung besteht üblicherweise darin, aus mehreren Alternativen eine auszuwählen. Dazu werden Bewertungskriterien oder Ziele definiert, und geprüft, wie gut jede Alternative diese Kriterien unterstützt. Wie man die Entscheidungen, die bei der Erfassung von Anforderungen nötig sind, treffen und methodisch unterstützen soll, war bisher keine explizite Forschungsfrage. Um konkrete Hilfestellung hierfür bieten zu können, werden in diesem Kapitel Erkenntnisse aus verschiedenen Bereichen zusammengefasst und angepasst. Wir gehen davon aus, dass bei diesen Entscheidungen am besten in zwei Schritten vorgegangen wird: Erstens die Identifikation möglicher Anforderungen (s. Kapitel 2.1) und zweitens die Beurteilung, ob diese Anforderung in die Spezifikation aufgenommen wird oder nicht (s. Kapitel 2.2). Hinter dieser Aufteilung steckt das Wissen, dass binäre ja-nein-Fragen leichter zu treffen sind als offene Fragen im Stil von: „Welche Anforderungen soll das System erfüllen?“

2.1 Unterstützung der Identifikation möglicher Anforderungen

RE Methoden für die Identifikation möglicher Anforderungen unterstützen gewöhnlich die oben genannte Umwandlung offener Fragen in überschaubare ja-nein-Entscheidungen. Prototyping z.B. erstellt einen vorläufigen Vorschlag, der es den Stakeholdern erlaubt, zu entscheiden: „Ja, so soll es sein“ oder „Nein, so nicht.“ Man spricht auch von IKIWISI (I’ll Know It When I See It) [Boe00].

Entscheidungen können von einander abhängen, beispielsweise wenn das Ergebnis einer Entscheidung beeinflusst, welche Alternativen bei einer anderen Entscheidung zur Verfügung stehen oder wie gut die Alternativen das Entscheidungskriterium erfüllen. Um trotz solcher Abhängigkeiten die besten Entscheidungen zu treffen, gilt es, die optimale Reihenfolge zu finden. Robinson *et al.* [RPV03] empfehlen die Priorisierung von Entscheidungen. Rittel und Webber [RW84] sprechen jedoch auch vom “wicked problem”: Versucht man, alle Entscheidungen systematisch zu treffen, führt das oft zum Deadlock. Üblicherweise trifft man Entscheidung vom Groben zum Feinen, d.h. diejenigen mit den am weitesten reichenden Auswirkungen zuerst (siehe für den Software-Entwurf [BD04]). Die Darstellung in einem Entscheidungsbaum ist sowohl üblich als auch hilfreich: „… a causal graph. A causal view of design decisions is a directed acyclic graph where each node is a design decision, the parents of a node are those decisions that constrained it.” [BCNS06] Analog gehen auch viele RE Methoden hierarchisch vom Groben zum Feinen vor und dokumentieren Anforderungen sowie die getroffenen Entscheidungen in Form eines Baums. Die zu treffenden Entscheidungen, an deren Ende die Anforderungsspezifikation steht, werden so in überschaubare Einzelentscheidungen zerlegt. Beispiele für solche Methoden sind TORE (=Task-Oriented Requirements Engineering) [PK03], das NFR Framework [CNYM00], die IESE NFR-Methode [KDP04, DKT05] und die zielorientierten RE Methoden [vL01].

Diese Entscheidungen werden weiter erleichtert durch die Wiederverwendung von Wissen, z.B. in Form von Fragen und deren Lösungs-Alternativen [HP06]. Dann wird die Frage: „Welches Y verfeinert Z?“ vereinfacht in die ja-nein-Frage: „Verfeinert das auf der Checkliste genannte Y hier Z oder nicht?“ Wie Firesmith [Fi04] beobachtet, sind gerade NFA besonders oft wieder verwendbar. Daher ist es folgerichtig, dass gerade NFA-Methoden wieder verwendbare Inhalte anbieten, z.B. als Checklisten [BSI91], Vorlagen [Fi04], Qualitätsmodelle [KDP04, DKT05] oder Rationale [DMM06].

2.2 Entscheidungskriterien

Prioritäten von Anforderungen werden an vielen Stellen des Software Engineering als Entscheidungskriterien verwendet. Das folgende Beispiel weist jedoch darauf hin, dass bei verschiedenen Entscheidungen unterschiedliche Kriterien eine Rolle spielen: Üblicherweise führen Volatilität und andere mit einer Anforderung verbundenen Risiken zur Verminderung deren Priorität [HPP06, XMCD04], das heißt, sie wird dann eher nicht umgesetzt. Nun priorisiert jedoch eine Arbeit [SW04] die Anforderungen *nach* deren Implementierung im Hinblick auf die Frage, welche Anforderungen besonders gründlich getestet werden sollen, und hier erhöht die Volatilität einer Anforderung deren Priorität, was Sinn ergibt. Das richtige Priorisierungskriterium hängt also von der Frage ab, die es beantworten soll und kann nicht bedenkenlos für verschiedene Entscheidungen verwendet werden.

Anhand welches Kriteriums könnte man entscheiden, welche Anforderung wichtig genug ist, um aufgeschrieben zu werden? Üblicherweise misst ein Entscheidungskriterium Zufriedenheit [YT97] oder die Erreichung anderer Ziele. Die in der Entscheidungstheorie [PRS95], [RRM02] üblichsten Kriterien sind der erwartete monetäre Wert, die Desirability (d.h. die subjektive emotionale Erwünschtheit) und die Utility, welche die Risikobereitschaft des Entscheiders mitberücksichtigt. Für die Priorisierung von Anforderungen innerhalb einer Liste gibt es eine Vielfalt möglicher Kriterien, wobei wir absehen von denen, die erst bei Kenntnis der technischen Umsetzung beurteilt werden können (z.B. Kosten [KAK01, IKO01, KR97], Kalender-Zeit für die Realisierung [REP03, WA06], u.v.m.), da sie noch nicht abschätzbar sind bevor die Anforderungen spezifiziert und analysiert wurden. Aus Stakeholdersicht bzw. bereits während der Erfassung bewertbar sind die folgenden Kriterien:

- Nutzen für die Stakeholder [GBB06] oder Stakeholderzufriedenheit [KR97,RR99]
- Erreichung anderer Ziele [CNYM00,BBH95]
- Bedeutung der Quelle / des Stakeholders, dem die Anforderung wichtig ist [WA06]
- Bietet die Konkurrenz diese Anforderung an oder nicht? [WA06]
- Risiko, d.h. durch die Realisierung der Anforderungen verursachter möglicher Schaden und dessen Wahrscheinlichkeit [PPBI99, Ru04], Volatilität [WA06]
- Dringlichkeit [Ru04], Unzufriedenheit bei Nicht-Realisierung [RR99], [Ru04]

Es ist auch möglich, mehrere Kriterien kombiniert zu berücksichtigen, beispielsweise als Nutzen-Kosten-Verhältnis oder Nettowert (Nutzen minus Kosten), wie es das SQUARE Project [XMCD04] und CBAM [KAK01, IKO01] tun, als eine gewichtete Summe oder als jede beliebige Funktion, die maximiert oder minimiert werden soll.

Bei der Entscheidung, welche Anforderungen aufgeschrieben werden sollen, können durchaus andere Kriterien eine Rolle spielen als bei deren späteren Priorisierung. Das Ziel der Erfassung kann sein, so viele Anforderungen aufzuschreiben, dass die Bedürfnisse der Stakeholder vollständig abgedeckt werden. Zu viele Anforderungen aufzuschreiben bedeutet eventuell weniger ein Problem als etwas Wichtiges zu vergessen, da überflüssige oder unrealistische Anforderungen später wieder aussortiert werden, fehlende aber vielleicht nicht mehr entdeckt. Es kann aber auch das Ziel sein, nur die wichtigsten Anforderungen zu erfassen, die „trivialen" jedoch wegzulassen. In einem sicherheitskritischen Bereich wird man gründlicher vorgehen als bei einer Software, die fast risikolos auch als Prototyp zum Einsatz kommen kann. Grundsätzlich sind die gleichen Kriterien denkbar wie sie oben für die Priorisierung aufgezählt wurden. Bewährt hat es sich insbesondere, bei der Spezifikation auf diejenigen Anforderungen zu fokussieren, die ein höheres Ziel erfüllen. Daher stehen bei Methoden, die hierarchisch Anforderungen durch Verfeinerung herleiten, an der Spitze oft Ziele. Dies gilt nicht nur für das zielorientierte RE [vL01]), sondern Ziele können wie in TORE [PK03] auch die zu unterstützenden Aufgaben sein. Erwähnt werden soll hier eine Arbeit [PW04], die behauptet, dass nur funktionale Anforderungen (FA) primäre Anforderungen sind, die dem Kunden Nutzen bringen, alle NFA sowie weitere FA dagegen unterstützen und beschränken die primären und werden aus diesen hergeleitet.

3 MOQARE

MOQARE (=Misuse-oriented Requirements Engineering) [HP05] ist eine Methode, mit der NFA erfasst und dokumentiert werden. MOQARE weist die im vorigen Kapitel empfohlenen Eigenschaften auf. Ausgehend von Zielen werden durch hierarchische Verfeinerung und mit Hilfe von Checklisten diejenigen NFA identifiziert, die besonders relevant sind. Bei der Verfeinerung werden hier nicht nur erwünschte Eigenschaften, sondern als Zwischenschritt auch unerwünschte erfasst, beispielsweise mögliche Geschäftsschäden. Es hat sich gezeigt, dass gerade diese unerwünschten Eigenschaften geeignet sind, um Begründungen hinter Anforderungen zu dokumentieren, und um die Anforderungen zu priorisieren. Da jede Anforderung von den Geschäftszielen herleitbar sein muss, können keine NFA erfasst werden, die nicht zumindest indirekt mindestens ein Geschäftsziel unterstützen. Das Ergebnis ist der Misuse Tree, eine grafische Übersichtsdarstellung der Anforderungen und deren Herleitung (s. Abbildung 1).

MOQARE bietet für den Entscheidungsprozess eine Schritt-für-Schritt-Unterstützung. Die zu treffenden Entscheidungen bei der Ermittelung der MOQARE-Elemente sind ganz klar definiert, ihre Ergebnisse hängen jedoch vom speziellen System ab und verlangen Domänen- und oft auch Technikwissen. Für die Schritte von MOQARE sind die zu beantwortenden Fragen, Checklisten und Kriterien definiert:

- Welches sind die wichtigsten Geschäftsziele, die das System unterstützt? Sie gehören zu fünf Dimensionen [Wi02]: product size, quality, staff, cost, (calendar) time. Die Auswahlkriterien für die Geschäftsziele definieren die Stakeholder. Ein sehr häufiges Geschäftsziel ist „Gewinn".
- Welche Geschäftsschäden bedrohen dieses Geschäftsziel zu einem bedeutenden Anteil? „Bedeutend" hängt davon ab, welches Risiko man einzugehen bereit ist.

– Welche Qualitätsmängel des IT-Systems führen in besonderem Maße zu diesen Geschäftsschäden? Welche Qualitätsattribute sind folglich besonders wichtig? Als Checkliste für die Qualitätsattribute dient die ISO 9126 [ISO91].
– In Bezug auf welchen Wert sollen diese Qualitätsattribute besonders erfüllt werden? Ein Wert kann jeder Teil des Systems sein, nicht nur Software, Hardware, Daten und Netzwerk, sondern auch Gebäude, Firma und Personal.
– Welche Bedrohungen können das Qualitätsziel (d.h. die Kombination aus Wert und Qualitätsattribut) mit besonders hohem Risiko bedrohen? Die Checklisten enthalten detaillierte Bedrohungen pro Qualitätsattribut. Das Risiko ist definiert als das Produkt aus Wahrscheinlichkeit und verursachtem Schaden [ISO02].
– Wer sind Misuser, die diese Bedrohung mit hoher Wahrscheinlichkeit ausführen? Misuser sind Personen, Rollen, Gruppen, Teile des IT-Systems oder seiner Umgebung sowie Naturgewalten wie Feuer oder Gewitter.
– Welche Schwachstellen können diese Bedrohungen ermöglichen, deutlich verschlimmern oder sogar provozieren? MOQARE bietet auch Checklisten, auf denen pro Wert bekannte Schwachstellen gelistet sind.
– Mit welchen Gegenmaßnahmen kann man eine Bedrohung verhindern, ihre Wahrscheinlichkeit oder verursachten Schaden verringern, oder sie zumindest entdecken? Gegenmaßnahmen sind neue FA, erweiterte FA, NFA an FA, NFA an die Architektur, die Benutzeroberfläche, das Projekt und die Softwareentwicklung, NFA an die Wartung oder neue Qualitätsziele. Die Checklisten schlagen Gegenmaßnahmen vor, passend zur Bedrohung oder zur Schwachstelle.

Ist diese Gegenmaßnahme konkret genug, um sie zu realisieren und zu prüfen oder stellt sie eher ein Qualitätsziel dar? Handelt es sich um ein neues Qualitätsziel, muss an dieser Stelle weiter analysiert werden.

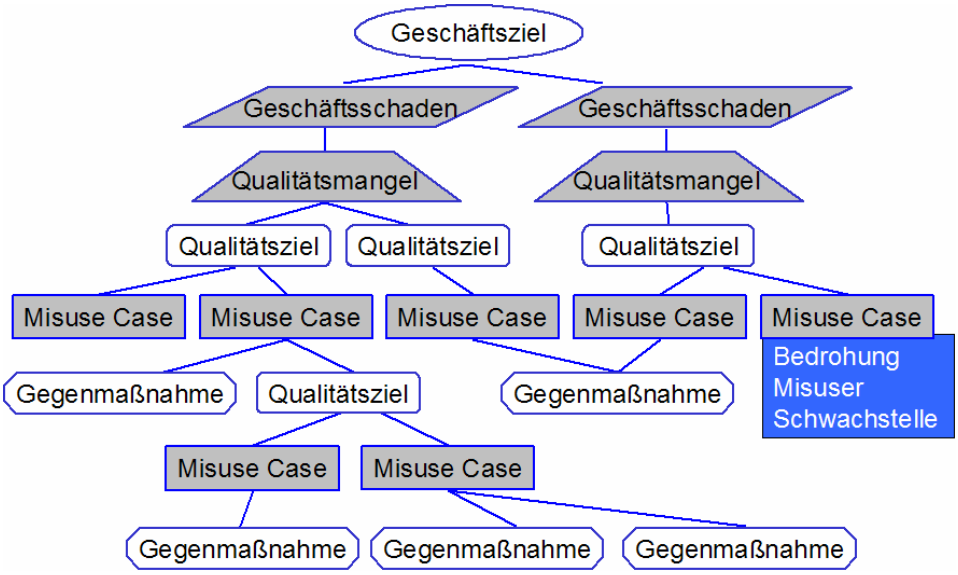

Abbildung 1: MOQARE Misuse Tree

4 Zusammenfassung

Dieser Beitrag behandelt die Entscheidungen, die beim Erfassen von NFA zu treffen sind. Hierbei werden in einem ersten Schritt mögliche Anforderungen identifiziert und in einem zweiten entschieden, ob sie relevant genug sind, um aufgeschrieben zu werden. Es wird dargestellt, wie diese Entscheidungen durch RE-Methoden unterstützt werden können und wie sie bisher schon unterstützt werden, beispielsweise durch MOQARE.

Literaturverzeichnis

[BCNS06] Bass, L.; Clements, P.; Nord, R.L.; Stafford, J.: Capturing and Using Rationale for a Software Architecture. In (Dutoit, A.H.; McCall, R.; Mistrík, I.; Paech, B., Hrsg.): Rationale Management in Software Engineering. Springer-Verlag, 2006

[BBH95] Boehm, B.W.; Bose, P.; Horowitz, E.; Lee, M.J.: Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach. In: Proc. 17th Int. Conf. on Software Eng. ICSE, 1995; S. 243-253.

[BD04] Bruegge, B.; Dutoit, A.H.: Object-Oriented Software Engineering - Using UML, Patterns, and Java. Prentice Hall, NJ, 2004.

[Boe00] Boehm, B.: Requirements that Handle IKIWISI, COTS, and Rapid Change. In: IEEE Computer 33(7) 2000; S. 99–102.

[BSI91] BSI (Bundesamt für Sicherheit in der Informationstechnik = German Ministery for Security in Information Technology): Information Technology Security Evaluation Criteria. http://www.bsi.bund.de/zertifiz/itkrit/itsec-en.pdf, 1991

[CNYM00] Chung, L.; Nixon, B.A.; Yu, E.; Mylopoulos, J.: Non-Functional Requirements in Software Engineering, Kluwer Acadamic Publishers, 2000.

[DKT05] Doerr, J.; Kerkow, D.; Koenig, T.; Olsson, T.; Suzuki, T.: Non-Functional Requirements in Industry - Three Case Studies Adopting an Experience-based NFR Method. In: Proc.s 13th IEEE Int. Conf. on Requirements Eng., 2005; S. 373 - 384

[DMM06] Dutoit, A.H.; McCall, R.; Mistrik, I.; Paech, B.: Rationale Management in Software Engineering, Springer-Verlag/Computer Science Editorial, 2006

[Fi04] Firesmith, D.G.: Specifying Reusable Security Requirements. In: J. of Object Technology, 3(1) 2004; S. 61-75.

[GBB06]Grünbacher, P.; Boehm, B.W.; Briggs, R.O.: Easy WinWin. http://sunset.usc.edu/research/WINWIN/EasyWinWin/ (last visited: dec 2006)

[HP05] Herrmann, A.; Paech, B.: Quality Misuse. In: Proc. Of REFSQ - Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality, Essener Informatik Berichte, 2005; S. 193-199.

[HP06] Herrmann, A.; Paech, B.: Lernen aus dokumentierten Architektur-Entscheidungen. In: Softwaretechnik-Trends 26(4) 2006; S. 22-27.

[HPP06] Herrmann, A.; Paech, B., Plaza, D.: ICRAD: An Integrated Process for Requirements Conflict Solution and Architectural Design. Int. J. of Software Eng. and Knowledge Eng. 16(6) 2006; S. 1-34.

[IKO01] In, H.; Kazman, R.; Olson, D.: From Requirements Negotiation to Software Architectural Decisions. In: Proc. From Software Requirements to Architectures Workshop STRAW, 2001

[ISO02] International Standards Organization, ISO: Risk management – Vocabulary – Guidelines for use in standards, ISO Guide 73, International Standards Organization, Geneva, 2002

[ISO91] International Standard ISO/IEC 9126. Information technology - Software product evaluation - Quality characteristics and guidelines for their use.

[KAK01]Kazman, R.; Asundi, J.; Klein, M.: Quantifying the Cost and Benefits of Architectural Decisions. In: Proc. Int. Conf. Software Eng., 2001; S. 297-306.

[KDP04] Kerkow, D.; Doerr, J.; Paech, B.; Olsson, T.; Koenig, T.: Elicitation and Documentation of Non-functional Requirements for Sociotechnical Systems. In (Maté, Silva, Hrsg.): Requirements Engineering for Sociotechnical Systems, Idea Group, 2004

[KR97] Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software 14(5) 1997; S. 67–74.

[PK03] Paech, B.; Kohler, K.: Task-driven Requirements in object-oriented Development. In (Leite, J.; Doorn, J., Hrsg.): Perspectives on Requirements Engineering, Kluwer Academic Publishers, 2003

[PPBI99] Park, J.; Port, D.; Boehm, B.; In, H.: Supporting Distributed Collaborative Prioritization for WinWin Requirements Capture and Negotiations. In: Prof. of Int. 3rd World Multiconf. on Systemics, Cybernetics and Informatics, Vol.2, 1999; S. 578-584.

[PW04] Poort, E.R.; de With, P.H.N.: Resolving Requirement Conflicts through Non-Functional Decomposition. In: Proc. of. 4th Workshop on Software Architecture, 2004; S. 145-154.

[REP03] Ruhe, G.; Eberlein, A.; Pfahl, D.: Trade-Off Analysis For Requirements Selection. Int. J. of Software Eng. and Knowledge Eng. 13 (4) 2003; S. 345-366.

[RPA01] Regnell, B.; Paech, B.; Aurum, A.; Wohlin, C.; Dutoit A.; Natt och Dag, J.: Requirements Mean Decisions! – Research issues for understanding and supporting decision-making in Requirements Engineering. In: Proc. 1st Swedish Conf. on Software Eng. Research and Practice (SERP'01), 2001; S. 49-52.

[RPV03] Robinson, W. N.; Pawlowski, S. D. ; Volkov, V.. Requirements Interaction Management, ACM Computing Surveys 35(2) 2003; S. 132–190.

[RR99] Robertson, S.; Robertson, J.: Mastering the Requirements process. Addison-Wesley, 1999.

[PRS95] Pratt, J.W.; Raiffa, H.; Schlaifer, R.: Introduction to Statistical Decision Theory. MIT Press, Cambridge, London, 1995

[RRM02] Raiffa, H.; Richardson, J.; Metcalfe, D.: Negotiation analysis - the science and art of collaborative decision making. Belknap, Cambridge, 2002.

[Ru04] Rupp, C.: Requirements-Engineering und –Management - Professionelle, iterative Anforderungsanalyse für die Praxis. Carl Hanser Verlag München Wien, 2004.

[RW84] Rittel, H.; Webber, M.: Planning problems are wicked problems. In (Cross N., Hrsg.): Development in Design Methodology. Wiley, Chichester, UK, 1984; S. 135-144.

[SW04] Srikanth, H.; Williams, L.: Requirements Based Test Case Prioritization. In: Student Research Forum, 12th ACM SIGSOFT Int. Symp. on Foundations of Softw. Eng., 2004

[vL01] van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Prof. of 5th Int. Symposium on Requirements Eng., 2001; S. 249-263.

[WA06] Wohlin, C.; Aurum A.: Criteria for Selecting Software Requirements to Create Product Value: An Industrial Empirical Study". In (Biffl, S.): Value-Based Software Engineering, Springer, Berlin, Heidelberg, 2006.

[Wi02] Wiegers, K.E.: Success Criteria Breed Success, The Rational Edge, 2(2) 2002

[XMCD04] Xie, N.; Mead, N.R.; Chen, P.; Dean, M.; Lopez, L.; Ojoko-Adams, D.; Osman, H.: SQUARE Project: Cost/Benefit Analysis Framework for Information Security Improvement Projects in Small Companies, Technical Note

[YT97] Yen, J.; Tiao, W.A.: A Systematic Tradeoff Analysis for Conflicting Imprecise Requirements. In: Proc. 3rd IEEE Int. Symposium on Requirements Eng. 1997; S. 87-97.