Was eine Requirements Engineering Methode sonst noch kann

Dr. Andrea Herrmann
Universität Heidelberg, Fakultät für Mathematik und Informatik, Lehrstuhl Software Engineering Group,
Universität Heidelberg; herrmann@informatik.uni-heidelberg.de

Abstract: Unter „Requirements Engineering“ (RE) versteht man gewöhnlich die Erfassung, Dokumentation und Verwaltung von Anforderungen an ein IT-System. Es wäre in der Praxis nützlich, nicht nur die Ergebnisse des RE, sondern auch die Methode in weiteren Aktivitäten des Software Engineering zu verwenden. Wir haben unsere RE-Methode für nicht-funktionale Anforderungen zusätzlich zum RE bei der Qualitätssicherung von Anforderungsspezifikationen und von IT-Systemen, im Wissensmanagement und für die Releaseplanung eingesetzt. Anhand dieser Erfahrungen diskutiert dieser Beitrag, welche der Eigenschaften der RE-Methode diesen über das RE hinaus gehenden Einsatz unterstützten, welchen Nutzen dies brachte und wo die Grenzen lagen. Diese Überlegungen werden dann auf andere RE-Methoden verallgemeinert.

Motivation

In der Praxis des Software Engineering verwendet man meist für die verschiedenen Aktivitäten jeweils ein anderes Werkzeug. Insbesondere für rein textuelle oder grafische Beschreibung der Anforderungen fehlt oft auch deren Integration. Das macht das Arbeiten mühsam, Dokumente werden inkonsistent und irgendwann gar nicht mehr gepflegt. Die Ursache dieses Problems ist jedoch weniger eine technische als eine methodische. Denn für die verschiedenen Aktivitäten werden auch unterschiedliche Methoden eingesetzt. Darum wäre es nützlich, eine Methode und deren Ergebnisse übergreifend für mehrere Software Engineering Aktivitäten verwenden zu können.

Wir haben MOQARE (Misuse-oriented Requirements Engineering), unsere RE-Methode für nicht-funktionale Anforderungen, zusätzlich zum RE (einschließlich Prozessanalyse) auch bei der Qualitätssicherung von Anforderungs-spezifikationen und von IT-Systemen, Wissensmanagement in Bezug auf nicht-funktionale Anforderungen und Releaseplanung nutzbringend eingesetzt. Gerade Qualitätssicherung und Wissensmanagement sind Stiefkinder in der Praxis des Software Engineering, deren Nutzen für die Risikominimierung zwar nicht zu leugnen ist, die jedoch keinen kurzfristif sichtbaren, konstruktiven Beitrag zum Projektprodukt (Software und Hardware) leisten und einen hohen Aufwand verursachen. Gleichzeitig sind sie naturgemäß eng mit dem RE verbunden, da die Anforderungen die Grundlage der Entwicklung des IT-Systems darstellen. Daher müssen die Anforderungen selbst von zuverlässiger Qualität sein, idealerweise in ähnlichen Projekten wieder verwendbar [1] und sind der Maßstab, an dem die Qualität des IT-Systems gemessen wird [2].

Ausgehend von unseren Erfahrungen mit MOQARE diskutieren wir, welche seiner Eigenschaften seinen breiten Einsatz unterstützten, welchen Nutzen dies brachte und wo die Grenzen lagen.

Eigenschaften von MOQARE

MOQARE identifiziert und dokumentiert nicht-funktionale Anforderungen und konkretisiert sie vom Groben zum Feinen, und zwar von Geschäftszielen (z.B. „Profit“) bis hinunter zu konkreten Anforderungen, die hier „Gegenmaßnahmen“ genannt werden

(z.B. „Auf Benutzeroberfläche X12 sind nur ‚Größe´ und das ‚Gewicht´ Pflichtfelder.“), siehe Abbildung 1. Im Folgenden werden diejenigen Eigenschaften genannt, die die breite Anwendung von MOQARE unterstützt haben. Eine ausführliche Methoden-beschreibung ist anderswo nachzulesen [3-5].

Unsere Erfahrungen

Unter „Requirements Engineering“ (RE) fasst man gewöhnlich die Erfassung, Dokumentation, Analyse und Verwaltung von Anforderungen an ein IT-System zusammen. Die Methode MOQARE wurde ursprünglich für das RE von nicht-funktionalen Anforderungen entwickelt. Wir haben sie aber auch explorativ in inzwischen einem Dutzend Fallstudien und Beispielen bei anderen Aktivitäten nutzbringend eingesetzt. Im Folgenden nennen wir nur wenige typische Beispiele pro Aktivität und welche Eigenschaften von MOQARE sich jeweils als nützlich erwiesen haben.

Erfassung, Dokumentation und Verwaltung nicht-funktionaler Anforderungen:

In der Uveitis-Fallstudie wurden nachträglich die nicht-funktionalen Anforderungen einer Patienten- und Befunddatenbank erfasst, nachdem die erste Version bereits im operativen Betrieb war [6]. Die Herleitung von Anforderungen aus Geschäftszielen stellt sicher, dass nur diejenigen Anforderungen erfasst werden, die – direkt oder auch indirekt – Geschäftsziele unterstützen. In der Uveitis-Fallstudie lauteten die Geschäftsziele „effiziente und gute Behandlung der Patienten“ sowie „gute wissenschaftliche Studie“. Es wurden Gegenmaßnahmen spezifiziert wie beispielsweise „Plausibilitätsprüfung der manuell eingegebenen Daten“. Die übersichtliche, hierarchische grafische Darstellung der MOQARE-Ergebnisse sowie deren Abhängigkeiten im Misuse Tree erwies sich bei der Erfassung der Anforderungen als **Interview-Leitfaden**, der sowohl die noch zu beantwortenden Fragen definiert und damit den Erfassungsprozess strukturiert als auch ein klares Ende vorgibt (siehe Vollständigkeitsbedingungen weiter unten). Außer Anforderungen an das IT-System wurden auch Anforderungen an das Umfeld, den Betrieb und den Entwicklungsprozess definiert. Es wurde dann geprüft, welche der Anforderungen bereits umgesetzt waren.

Die Anwendung von MOQARE erlaubte so eine umfassende und zielgerichtete Erfassung der Anforderungen. Durch die Priorisierung der nicht realisierten Gegenmaßnahmen konnten diese den **Versionen für die Weiterentwicklung** des Systems zugeordnet werden.

Wir dokumentierten im Misuse Tree aber auch in mehreren Fällen **Anforderungen an einen Prozess** und an dessen Werkzeugunterstützung. In einem Fall wurde die Prozessanalyse über weitere Misuse Tree Ebenen bis zu technischen Anforderungen weitergeführt. Dies ist möglich, da MOQARE auf Use Cases und Misuse Cases [1,7,8] basiert. Diese beschreiben Schritt für Schritt die Wechselwirkung zwischen System und Aktor, können aber auch Prozess-anforderungen oder technische Anforderungen darstellen, je nach dem, wie man „System“ und „Aktor“ definiert. Diese breite Verwendbarkeit vererbt sich auf MOQARE.

Qualitätssicherung von Anforderungs-dokumenten:

In drei Fällen wurden fertige umfangreiche Spezifikationsdokumente in die Form eines Misuse Tree gebracht. Die Dokumente bestanden stets aus einer Sammlung von Freitext und Tabellen. Beim ersten handelte es sich um Dokumente aus einem realen IT-Projekt (die 160-Seiten-Spezifikation der Kundenanforderungen und die technische Spezifikation von 240 Seiten), ein Mal um ein fiktives Fallbeispiel (40 Seiten) im Rahmen eines Forschungsprojektes und das dritte Dokument war das 50-seitige Ergebnis der Analyse der BASEL II Anforderungen des GRIF Projects [9,10].

Anforderungen sollen laut IEEE Standard [11] die folgenden Eigenschaften haben: korrekt; eindeutig; vollständig; konsistent; bewertet nach Wichtigkeit oder Stabilität;

prüfbar; modifizierbar; nachverfolgbar. MOQARE unterstützt besonders die Erreichung und Inspektion von Vollständigkeit, Konsistenz, Priorisierung, Prüfbarkeit und Nachverfolgbarkeit:

Vollständigkeit: Das Ende der MOQARE-Analyse ist dadurch definiert, dass die Blätter des Misuse Tree nur Gegenmaßnahmen sein dürfen, oder aber Qualitätsziele, die in einen anderen Verantwortungsbereich fallen (z.B. interner oder externer Dienstleiter, Hersteller einer verwendeten Standardsoftware), für die es Standardlösungen gibt, die als genügend erfüllt erachtet werden oder die eine geringe Priorität haben. Die Anzahl der herzuleitenden Gegenmaßnahmen bleibt Ermessenssache. Daher garantieren diese weichen Vollständigkeitsbedingungen die Vollständigkeit der Anforderungen nicht, verbessern sie jedoch. Beispielsweise wurde im Misuse Tree klar sichtbar, wenn Geschäftsziele oder Qualitätsziele gar nicht oder schlechter durch Gegenmaßnahmen unterstützt wurden als andere.

Konsistenz: Die Inspektion der Anforderungen hinsichtlich Konsistenz wird durch klare Definitionen der Konzepte und durch die grafische Übersichtsdarstellung im Misuse Tree unterstützt. Eine Erleichterung kann zusätzlich eine Toolunterstützung bieten, bei uns durch die Verlinkungen der MOQARE-Konzepte im webbasierten Sysiphus [12]. Im ersten Fallbeispiel wurden Inkonsistenzen zwischen den beiden Dokumenten entdeckt in Bezug auf Terminologie, Inhalt und Beschreibungsebene.

Priorisierung: MOQARE unterstützt die Priorisierung von Anforderungen (s. unten).

Prüfbarkeit: Dadurch dass zwischen (vagen) Qualitätszielen und Gegenmaßnahmen (=konkreten Anforderungen) unterschieden wird und ausdrücklich Gegenmaßnahmen hergeleitet werden sollen, ist die Prüfbarkeit der Anforderungen hoch. Denn Qualitätsziele bestehen in MOQARE aus einem Qualitätsattribut [13] und einem Wert (wie: „Benutzerfreundlichkeit des Systems“) und sind in dieser Form nicht objektiv prüfbar. Gegenmaßnahmen sind nach unserer Erfahrung funktionale Anforderungen oder nicht-funktionale Anforderungen an funktionale, außerdem nicht-funktionale Anforderungen an die Architektur, das Umfeld, den Betrieb und den Entwicklungsprozess.

Nachverfolgbarkeit: Die Anforderungen werden im Misuse Tree auf mehreren Ebenen beschrieben, die erwünschten Eigenschaften als Geschäftsziele, Qualitätsziele und Gegenmaßnahmen. Die Verfeinerungs-beziehungen zwischen diesen werden ebenfalls grafisch dokumentiert. Daher sind die Begründungen von Anforderungen nachvollziehbar und dokumentiert, bis hoch zu den Geschäftszielen der Firma oder der Kunden. In den ersten beiden Fallbeispielen fehlten Nachverfolgbarkeitsverbindungen zwischen diesen MOQARE-Ebenen. Beispielsweise war dem Geschäftsziel „Firmenimage“ das Qualitätsziel „hochklassige Technologie“ zugeordnet, allerdings blieb unklar, welche der Anforderungen dieses Ziel unterstützen.

Die Dokumentation nicht nur der erwünschten Eigenschaften, sondern auch der unerwünschten Effekte (Geschäftsschäden, Qualitätsmängel und Misuse Cases) ergänzt die Dokumentation von Begründungen hinter den Anforderungen. Im Gegensatz zu Use Cases, die die geplante Benutzung des Systems beschreiben, beschreiben Misuse Cases als Szenario absichtliche Angriffe, Unfälle und Benutzerfehler einschließlich der entstandenen Schäden. Das dritte der untersuchten Dokumente war das einzige, in dem eine Spezifikation alle MOQARE-Elemente enthielt, d.h. in diesem Sinne vollständig war. Dies lag daran, dass sie von Sicherheitsexperten erstellt wurde, die üblicherweise Anforderungen mit Misuse Cases begründen. Fehlende Begründungen sind sonst häufig, obwohl nach unserer Erfahrung der Großteil der Anforderungen durch solche Misuse Cases begründet ist.

Eine mehrstündige MOQARE-Analyse (ca. zehn Stunden in der ersten Fallstudie, zwei Stunden in den beiden anderen) konnte also die Qualität der Spezifikationsdokumente deutlich verbessern und bestätigen.

Qualitätssicherung eines IT-Systems:
Anforderungen sind der Maßstab, an dem die Qualität eines IT-Systems gemessen wird. Sie können in der Praxis für manuelle Systemtests umso eher als Testkriterien dienen, je detaillierter sie beschrieben sind, und insbesondere in Szenarien formulierte Anforderungen (z.B. Use Cases) können als rudimentäre Testfälle dienen. Sie müssen jedoch – insbesondere wenn sie die Grundlage für automatisierte Systemtests bilden – um zusätzliche Informationen ergänzt werden, beispielsweise um Testdaten (Input und Output) sowie Informationen über ihre Ausführungsreihenfolge.
In einer Bachelorarbeit [14] sollte die Benutzerfreundlichkeit einer **Software bewertet**, außerdem Verbesserungs-möglichkeiten identifiziert und priorisiert werden. Auf der Suche nach einer effizienten und leichtgewichtigen Methode hierfür griffen wir auf MOQARE zurück, um Gegenmaßnahmen herzuleiten, die als prüfbare Qualitätskriterien dienen konnten, ohne detaillierte Testfälle zu formulieren. Die Use Cases der Software dienten als Testfälle und bei deren manueller Durchführung wurde bewertet, wie gut das System jeweils die Qualitätskriterien erfüllt. Ein besonders wichtiges Kriterium lautete: „Alle nötigen Informationen müssen zur gleichen Zeit für den Benutzer verfügbar sein“. Was dieses Kriterium im Zusammenhang mit jedem Use Case jeweils bedeutete, konnte aus einer Prozessbeschreibung herausgelesen werden. Andere Qualitätskriterien verlangten eine subjektive Interpretation durch die beiden Tester, z.B.: „Intuitive Gestaltung der Durchführung der Aufgaben“. Solche allgemein formulierten und interpretationsbedürftigen Qualitätskriterien sind natürlich nur für manuelle Tests durch geübte Tester geeignet, dafür jedoch effizient. Die Ergebnisse der Tests wurden in einer Tabelle festgehalten, die die folgenden Auswertungen unterstützt: Basierend auf der quantitativen Bewertung des Nutzens der Gegenmaßnahmen (und der Use Cases) wurde die Gesamtqualität des Systems bewertet. Eine spätere Wiederbewertung der Software gegen dieselben Kriterien würde quantitativ die **Qualitätsverbesserung messen.** (Die Weiterentwicklung der Software steht momentan noch an.) Qualität ist in MOQARE definiert dadurch, dass Misuse Cases verhindert, entdeckt oder ihr Risiko vermindert wird. Ihr Risiko ist definiert als das Produkt aus der Wahrscheinlichkeit und dem verursachten Schaden. Der Nutzen einer Gegenmaßnahme und damit deren Priorität messen sich folglich daran, wie gut sie Risiken vermindern.

Wissensmanagement:
Die Wiederverwendung von Wissen in der Form von **Checklisten** hat sich in MOQARE bewährt, um nichts Wesentliches zu vergessen. Solche Checklisten haben wir für die beiden Teile der Qualitätsziele: die zu schützenden Werte und die Qualitätsattribute [5]. Beispielsweise bei der oben bereits genannten Uveitis-Fallstudie verhinderten die Checklisten, dass man sich – wie es in einem unstrukturierten Gespräch leicht geschieht – auf wenige Qualitätsattribute fokussiert, z.B. die benutzerbezogene Sicht. Die Checklisten brachten hier insbesondere den Aspekt der Weiterentwicklung des Systems mit hinein in Form der Qualitätsattribute Wartbarkeit und Portierbarkeit.
Die Abhängigkeiten und Begründungen stellen ebenfalls wieder verwendbares Wissen dar. So verwenden wir in MOQARE auch Checklisten für mögliche Schwachstellen von Werten sowie für Misuse Cases, die die Qualitätsattribute bedrohen, sowie Gegenmaßnahmen gegen diese Schwachstellen und Misuse Cases [5].
In einem Beispiel [15] haben wir statt den Anforderungen an eine konkrete Software ein **Qualitätsmodell** für die Interoperabilität von Web Services aufgestellt: Hierbei haben wir allgemeingültig analysiert, welche Gegen-maßnahmen zur Interoperabilität eines Web Service beitragen (beispielsweise verschiedene Standards). Dabei wurden auch Abhängigkeiten zu anderen Qualitäts-attributen dokumentiert, die zur Interoperabilität beitragen (z.B. Performanz,

Wartbarkeit und Portierbarkeit). Dieses Qualitätsmodell kann nun als Grundlage für die Spezifikation der nicht-funktionalen Anforderungen an einen konkreten Web Service dienen. Außerdem wurden dadurch, dass MOQARE die Anforderungen aus Geschäftszielen herleitet, auch die Interoperabilität begründet, d.h. es wurde dokumentiert, welche Geschäftsziele der Firma diese Qualität des Web Services unterstützt. Diese sind der hohe Marktanteil für den Hersteller und den Service-Broker sowie der einfache Einsatz durch den Kunden mit geringem Vorbereitungs- oder Wartungsaufwand.

Releaseplanung:
In zwei Fallstudien wurden nicht-funktionale Anforderungen an eine bereits existierende Software spezifiziert (Uveitis-Fallstudie und die Benutzerfreundlichkeitsbeurteilung, siehe oben). Anschließend lautete die nächste Frage, ob oder in welcher Reihenfolge man die nicht-realisierten Qualitätsanforderungen umsetzt, die ja **Verbesserungsmöglichkeiten** darstellen. Diese wurden nach ihrem Nutzen priorisiert und in Kombination mit einer Aufwandsschätzung den **Versionen für die Weiterentwicklung** der Software in Bezug auf die nicht-funktionalen Anforderungen zugeordnet.

Diskussion
Unsere Erfahrungen haben gezeigt, dass MOQARE außer im RE nicht-funktionaler Anforderungen besonders in der Qualitätssicherung von Anforderungs-spezifikationen und von IT-Systemen, Wissensmanagement und Releaseplanung effizient und zielführend eingesetzt werden kann. Wichtige Eigenschaften, die dies unterstützten waren die hierarchische Herleitung und Darstellung der Anforderungen, MOQAREs Vollständigkeitsbedingungen für die Anforderungen, die Wiederverwendung von Wissen und die Anforderungspriorisierung.
Die Qualitätssicherung von Anforderungsdokumenten wird unterstützt in Bezug auf Vollständigkeit, Konsistenz innerhalb eines Dokuments oder zwischen mehreren Dokumenten, Priorisierung, Prüfbarkeit und Nachverfolgbarkeit. Die durch MOQARE hergeleiteten Gegenmaßnahmen stellen zwar keine vollständigen Testfälle dar, können aber von erfahrenen Testern als Testkriterien für manuelle Systemtests verwendet werden. Die Checklisten erlauben die Wiederverwendung von Wissen bezüglich nicht-funktionaler Anforderungen über Projektgrenzen hinweg. Für die Releaseplanung schafft MOQARE eine Grundlage durch die Bewertung des Nutzens von Gegenmaßnahmen.
In allen diesen Anwendungsgebieten stellte MOQARE eine leichtgewichtige Methode dar, die effizient und nutzbringend in überschaubar großen Projekten eingesetzt werden kann. Sie stellt jedoch keinen vollwertigen Ersatz für spezialisierte Methoden dar.
Die breite Einsetzbarkeit von MOQARE würde sich noch verbessern, wenn die Methode in folgende Richtungen weiterentwickelt würde:

- Unterstützung der Prüfung von Anforderungen auf Korrektheit, Eindeutigkeit, Modifizierbarkeit
- Systematisch unterstützte Herleitung von Testfällen aus den Gegenmaßnahmen
- Releaseplanung durch Bewertung von Gegenmaßnahmen nicht nur nach Nutzen, sondern auch Kosten, sowie Sortierung der Gegenmaßnahmen nach Kriterien wie Nettowert = Nutzen minus Kosten.

Vergleich mit anderen Methoden
Die weit gefächerte Anwendbarkeit von MOQARE gilt sicher nicht nur für diese, sondern auch für andere RE-Methoden. Entsprechende Eigenschaften sind ja bei anderen Methoden ebenfalls vorhanden. Ohne einen umfassenden Methoden-Überblick geben zu wollen, nennen wir im Folgenden einige wichtige Methoden, welche ähnliche Eigenschaften wie MOQARE aufweisen und daher vermutlich auch ähnlich einsetzbar sind.
Eine hierarchische Darstellung von nicht-funktionalen Anforderungen, von oben nach unten verfeinert, bieten beispielsweise das

NFR Framework [16], die IESE NFR-Methode [17,18], zielorientierte RE-Methoden [19], u.v.m. [6]. Für funktionale Anforderungen wird diese Verfeinerung vom Groben zum Feinen beispielsweise durch TORE (=Task-oriented Requirements Engineering) [20] unterstützt.

Was MOQARE für die Anwendung in der Qualitätssicherung besonders auszeichnet sind die Vollständigkeitsbedingungen für die zu erfassenden Anforderungen. Solche sind bei anderen Methoden selten explizit (z.B. [17]).

Wissensmanagement durch wieder verwendbare Inhalte werden auch von anderen Methoden unterstützt, in verschiedensten Formen, z.B. als Checklisten [21], Patterns oder Vorlagen [1,18], Qualitätsmodellen [17,18] oder Rationale [22].

Die Priorisierung von Anforderungen wird von vielen Erfassungsmethoden gar nicht oder nur rudimentär unterstützt.

Ausblick

Die beherzte Anwendung von RE-Methoden in weiteren Software Engineering Aktivitäten möchten wir weiterempfehlen, insbesondere wenn es möglich ist, die unterstützende Software an die erweiterten Bedürfnisse anzupassen, und wenn Effizienz wichtiger ist als die perfekte Methode. Denn bei der Zweckentfremdung einer Methode in einem anderen Gebiet ist sie selten ein vollwertiger Ersatz für eine spezialisierte Methode.

Referenzen

[1] Donald G. Firesmith: Specifying Reusable Security Requirements. Journal of Object Technology, 3(1) January-February 2004, S. 61-75

[2] Dorothy Graham: Requirements and Testing: Seven Missing-Link Myths. IEEE Software 19(5), Sept/Oct 2002, S. 15-17

[3] Andrea Herrmann, Barbara Paech: MOQARE = Misuse-oriented Quality Requirements Engineering - Über den Nutzen von Bedrohungsszenarien beim RE von Qualitätsanforderungen. Softwaretechnik-Trends 26(1) Feb. 2006, S. 13-14

[4] Andrea Herrmann, Barbara Paech: Quality Misuse. REFSQ - Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality, Essener Informatik Berichte, 2005, S. 193-199

[5] Andrea Herrmann, Barbara Paech: Software Quality by Misuse Analysis. Technical Report SWEHD-TR-2005-01 (University of Heidelberg, 2005) http://www-swe.informatik.uni-heidelberg.de/research/publications/reports.htm

[6] Andrea Herrmann, Barbara Paech: MOQARE: Misuse-oriented Quality Requirements Engineering. Submitted to Requirements Engineering Journal

[7] Guttorm Sindre, Andreas L. Opdahl: Templates for Misuse Case Description. REFSQ - International Workshop on Requirements Engineering – Foundation for Software Quality, 2001, S. 125-136

[8] Ian Alexander: Initial Industrial Experience of Misuse Cases. Requirements Engineering Conference:, 2002, S. 61-68

[9] GRIF Project: “BASEL II Operational Risk Management Process Assessment Model”, Version 1.00, http://www.cssf.lu/docs/PAM_ORM_BALEII_V01_00.pdf

[10] GRIF Project: „BASEL II Operational Risk Management Process Reference Model”, Version 1.00, http://www.cssf.lu/docs/PRM_ORM_BALEII_V01_00.pdf

[11] IEEE: Std. 830-1998: IEEE Recommended Practice for Software Requirements Specification, 1998

[12] Sysiphus http://sysiphus.in.tum.de/, 2007

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

[14] unveröffentlicht, da teilweise vertraulich

[15] Andrea Herrmann, Jürgen Rückert, Barbara Paech: Exploring the Interoperability of Web Services using MOQARE. IS-TSPQ Workshop "First International Workshop on Interoperability Solutions to Trust, Security, Policies and QoS for Enhanced Enterprise Systems", 21. März 2006 in Bordeaux

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

[17] Daniel Kerkow, Jörg Doerr, Barbara Paech, Thomas Olsson, Tom Koenig: Elicitation and Documentation of Non-functional Requirements for Sociotechnical Systems. In: Maté and Silva (ed), Requirements Engineering for Sociotechnical Systems, Idea Group, Inc., 2004

[18] Jörg Doerr, Daniel Kerkow, Tom Koenig, Thomas Olsson, Takeshi Suzuki: Non-Functional Requirements in Industry - Three Case Studies Adopting an Experience-based NFR Method. Proceedings 13th IEEE International Conference on Requirements Engineering, 2005, S. 373 - 384

[19] A. van Lamsweerde: Goal-Oriented Requirements Engineering: A Guided Tour. 5. Internationales Symposium on Requirements Engineering, 2001, S. 249-263

[20] Barbara Paech, Kirstin Kohler: Task-driven Requirements in object-oriented Development.

Perspectives on Requirements Engineering, J. Leite and J. Doorn (Hrsg.), Kluwer Academic Publishers, 2003

[21] 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

[22] Allen H. Dutoit, Raymond McCall, Ivan Mistrik, Barbara Paech: Rationale Management in Software Engineering, Springer-Verlag/Computer Science Editorial, 2006

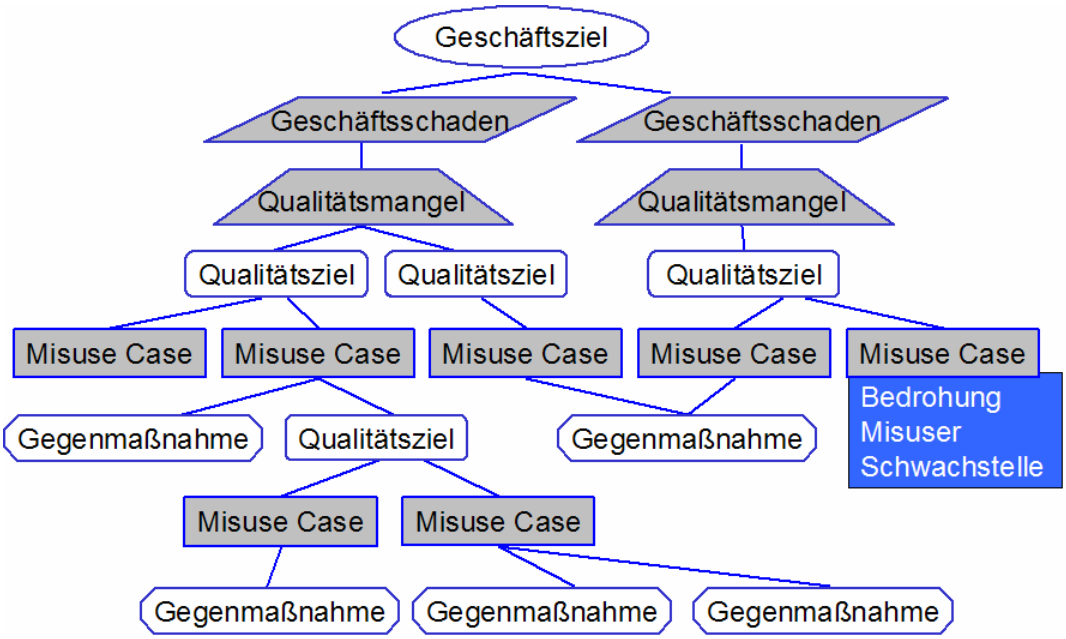

Abbildung 1: Misuse Tree